L Number	Hits	Search Text	DB	Time stamp
1	2892	345/744 OR 345/745 OR 345/747 OR 345/767 OR 345/765 OR 345/866 OR 345/762 OR 345/762 OR 709/331 OR 709/332 OR 709/328	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/31 15:40
2	2331	class ADJ objects	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/31 15:40
3	902	objects ADJ attributes	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/31 15:41
4	156	(345/744 OR 345/745 OR 345/747 OR 345/767 OR 345/765 OR 345/866 OR 345/762 OR 345/762 OR 709/331 OR 709/332 OR 709/328) and (class ADJ objects)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/31 15:41
5	13	(objects ADJ attributes) and ((345/744 OR 345/745 OR 345/747 OR 345/767 OR 345/765 OR 345/866 OR 345/762 OR 345/762 OR 709/331 OR 709/332 OR 709/328) and (class ADJ objects))	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/31 15:41
6	7	((objects ADJ attributes) and ((345/744 OR 345/745 OR 345/747 OR 345/767 OR 345/765 OR 345/866 OR 345/762 OR 345/762 OR 709/331 OR 709/332 OR 709/328) and (class ADJ objects))) and API	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/31 15:41
7	1	((objects ADJ attributes) and ((345/744 OR 345/745 OR 345/747 OR 345/767 OR 345/765 OR 345/866 OR 345/762 OR 345/762 OR 709/331 OR 709/332 OR 709/328) and (class ADJ objects))) and GUIs	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/31 15:42
8	6	(((objects ADJ attributes) and ((345/744 OR 345/745 OR 345/747 OR 345/767 OR 345/765 OR 345/866 OR 345/762 OR 345/762 OR 709/331 OR 709/332 OR 709/328) and (class ADJ objects))) and API) and properties	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/31 15:42
9	0	((((objects ADJ attributes) and ((345/744 OR 345/745 OR 345/747 OR 345/767 OR 345/765 OR 345/866 OR 345/762 OR 345/762 OR 709/331 OR 709/332 OR 709/328) and (class ADJ objects))) and API) and properties) and HTTP	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/31 15:42
10	0	(((objects ADJ attributes) and ((345/744 OR 345/745 OR 345/747 OR 345/767 OR 345/765 OR 345/866 OR 345/762 OR 345/762 OR 709/331 OR 709/332 OR 709/328) and (class ADJ objects))) and GUIs) and HTTP	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/31 15:43
11	0	(((objects ADJ attributes) and ((345/744 OR 345/745 OR 345/747 OR 345/767 OR 345/765 OR 345/866 OR 345/762 OR 345/762 OR 709/331 OR 709/332 OR 709/328) and (class ADJ objects))) and API) and HTTP	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/31 15:43
12	0	((objects ADJ attributes) and ((345/744 OR 345/745 OR 345/747 OR 345/767 OR 345/765 OR 345/866 OR 345/762 OR 345/762 OR 709/331 OR 709/332 OR 709/328) and (class ADJ objects))) and HTTP	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/31 15:44
13	124	(class ADJ objects) AND (objects ADJ attributes)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/31 15:44

14	46	((class ADJ objects) AND (objects ADJ attributes)) and API	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/31 15:45
15	46	(((class ADJ objects) AND (objects ADJ attributes)) and API) and attributes	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/31 15:45
16	40	((((class ADJ objects) AND (objects ADJ attributes)) and API) and attributes) and properties	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/31 15:45
17	11	(((((class ADJ objects) AND (objects ADJ attributes)) and API) and attributes) and properties) and HTTP	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/31 15:46
18	516	access ADJ attributes	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/31 15:47
19	0	((objects ADJ attributes) and ((345/744 OR 345/745 OR 345/747 OR 345/767 OR 345/765 OR 345/866 OR 345/762 OR 345/762 OR 709/331 OR 709/332 OR 709/328) and (class ADJ objects))) and (access ADJ attributes)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/31 15:47
20	15	((345/744 OR 345/745 OR 345/747 OR 345/767 OR 345/765 OR 345/866 OR 345/762 OR 345/762 OR 709/331 OR 709/332 OR 709/328) and (class ADJ objects)) and (access ADJ attributes)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/31 15:48
21	15	(((345/744 OR 345/745 OR 345/747 OR 345/767 OR 345/765 OR 345/866 OR 345/762 OR 345/762 OR 709/331 OR 709/332 OR 709/328) and (class ADJ objects)) and (access ADJ attributes)) and modify	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/31 15:49
22	887	predefined ADJ values	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/31 15:50
23	2	((class ADJ objects) AND (objects ADJ attributes)) and (predefined ADJ values)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/31 15:50
24	0	(((class ADJ objects) AND (objects ADJ attributes)) and (predefined ADJ values)) and compare	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/31 15:51
25	132	default ADJ interface	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/31 15:52
26	0	(((class ADJ objects) AND (objects ADJ attributes)) and (predefined ADJ values)) and (default ADJ interface)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/31 15:52

27	6	(((objects ADJ attributes) and ((345/744 OR 345/745 OR 345/747 OR 345/767 OR 345/765 OR 345/866 OR 345/762 OR 345/762 OR 709/331 OR 709/332 OR 709/328) and (class ADJ objects))) and API) and default	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/31 15:53
28	12646	unique ADJ identifier	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/31 15:53
29	3981	(unique ADJ identifier) and global	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/31 15:54
30	0	(((objects ADJ attributes) and ((345/744 OR 345/745 OR 345/747 OR 345/767 OR 345/765 OR 345/866 OR 345/762 OR 345/762 OR 709/331 OR 709/332 OR 709/328) and (class ADJ objects))) and API) and ((unique ADJ identifier) and global)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/31 15:55
31	0	((objects ADJ attributes) and ((345/744 OR 345/745 OR 345/747 OR 345/767 OR 345/765 OR 345/866 OR 345/762 OR 345/762 OR 709/331 OR 709/332 OR 709/328) and (class ADJ objects))) and ((unique ADJ identifier) and global)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/31 15:55
32	3	((objects ADJ attributes) and ((345/744 OR 345/745 OR 345/747 OR 345/767 OR 345/765 OR 345/866 OR 345/762 OR 345/762 OR 709/331 OR 709/332 OR 709/328) and (class ADJ objects))) and global	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/31 15:55
33	0	(unique ADJ identifier) and (((objects ADJ attributes) and ((345/744 OR 345/745 OR 345/747 OR 345/767 OR 345/765 OR 345/866 OR 345/762 OR 345/762 OR 709/331 OR 709/332 OR 709/328) and (class ADJ objects))) and global)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/31 15:56
34	1	(((objects ADJ attributes) and ((345/744 OR 345/745 OR 345/747 OR 345/767 OR 345/765 OR 345/866 OR 345/762 OR 345/762 OR 709/331 OR 709/332 OR 709/328) and (class ADJ objects))) and global) and identifier	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/31 15:56

predefined list and user defined value

US005982367A

United States Patent [19]

Alimpich et al.

[11] Patent Number: 5,982,367

[45] Date of Patent: Nov. 9, 1999

[54] **GRAPHICAL INTERFACE METHOD, APPARATUS AND APPLICATION FOR CREATING A LIST FROM PRE-DEFINED AND USER-DEFINED VALUES**

[75] Inventors: **Claudia C. Alimpich**; **Joan Stagaman Goddard**, both of Boulder; **Deborah E. Neuhard**, Longmont; **Cheryl Steinmeyer**, Franktown, all of Colo.; **Minh Trong Vo**, Mountain View, Calif.; **James Philip Wittig**, Boulder; **Rachel Youngran Yang**, Superior, both of Colo.

[73] Assignee: **International Business Machines**, Armonk, N.Y.

[21] Appl. No.: **09/097,448**

[22] Filed: **Jun. 15, 1998**

Related U.S. Application Data

[63] Continuation of application No. 08/696,752, Aug. 14, 1996.

[51] **Int. Cl.**[6] **G06F 3/00**

[52] **U.S. Cl.** **345/347**; 345/352

[58] **Field of Search** 345/326–358

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,287,514	2/1994	Gram	345/352
5,345,550	9/1994	Bloomfield	345/353
5,608,860	3/1997	Fitzpatrick et al.	345/352
5,625,781	4/1997	Cline et al.	345/335
5,666,502	9/1997	Capps	345/352
5,774,667	6/1998	Garvey et al.	345/333 X

OTHER PUBLICATIONS

Advanced Interface Design Guide, IBM Corp., pp. 40–41, 46–47, 95–97, Jun. 1989.

Cowart, Mastering Windows 3.1, Sybex Inc., pp. 189–193, 202,899, 1993.

Primary Examiner—John E. Breene

Attorney, Agent, or Firm—David W. Victor; Konrad Raynes & Victor

[57] **ABSTRACT**

A graphical user interface may used to construct a list from pre-defined and user-defined printer values. Predefined values may be added or removed through a dialog box by selecting or deselecting values. New user-defined values may added by filling in a value in the add value dialog. An existing value may be modified by selecting a modify push-button. In response to selecting the modify push-button, a value may be edited in a modify value dialog. Adding, removing, modifying, and deleting printer values may be performed by user interaction with various push-buttons and dialog boxes.

15 Claims, 9 Drawing Sheets

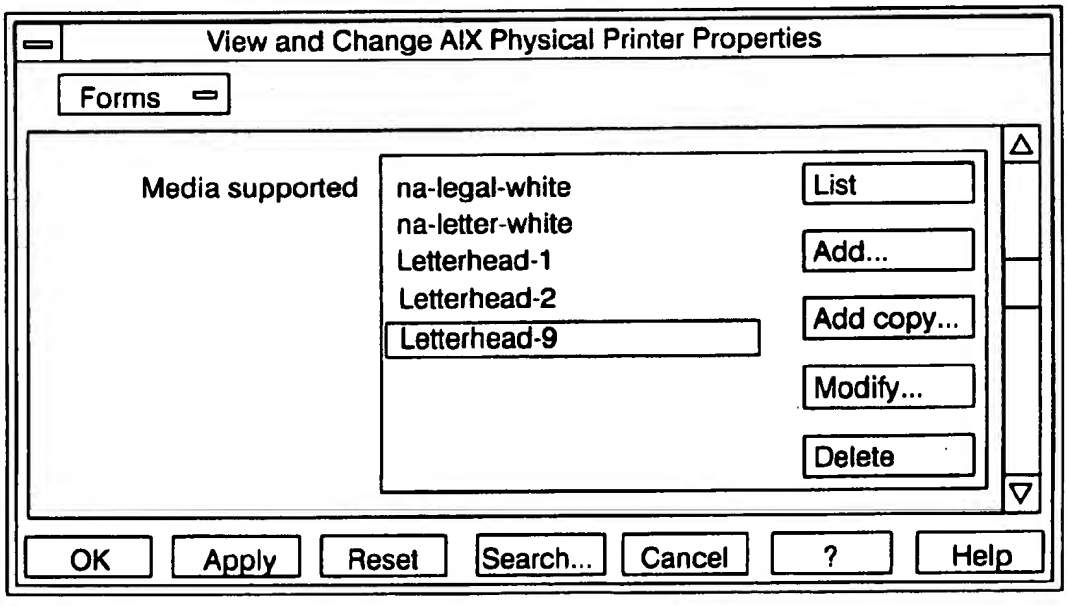
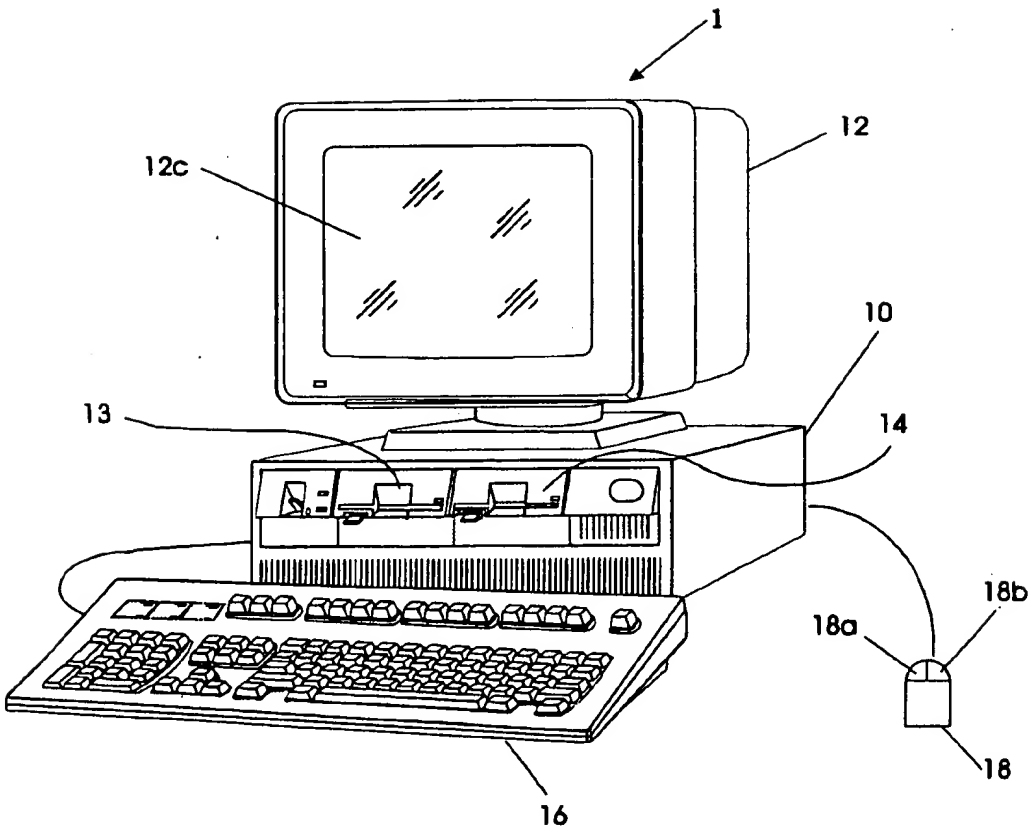


FIG. 1



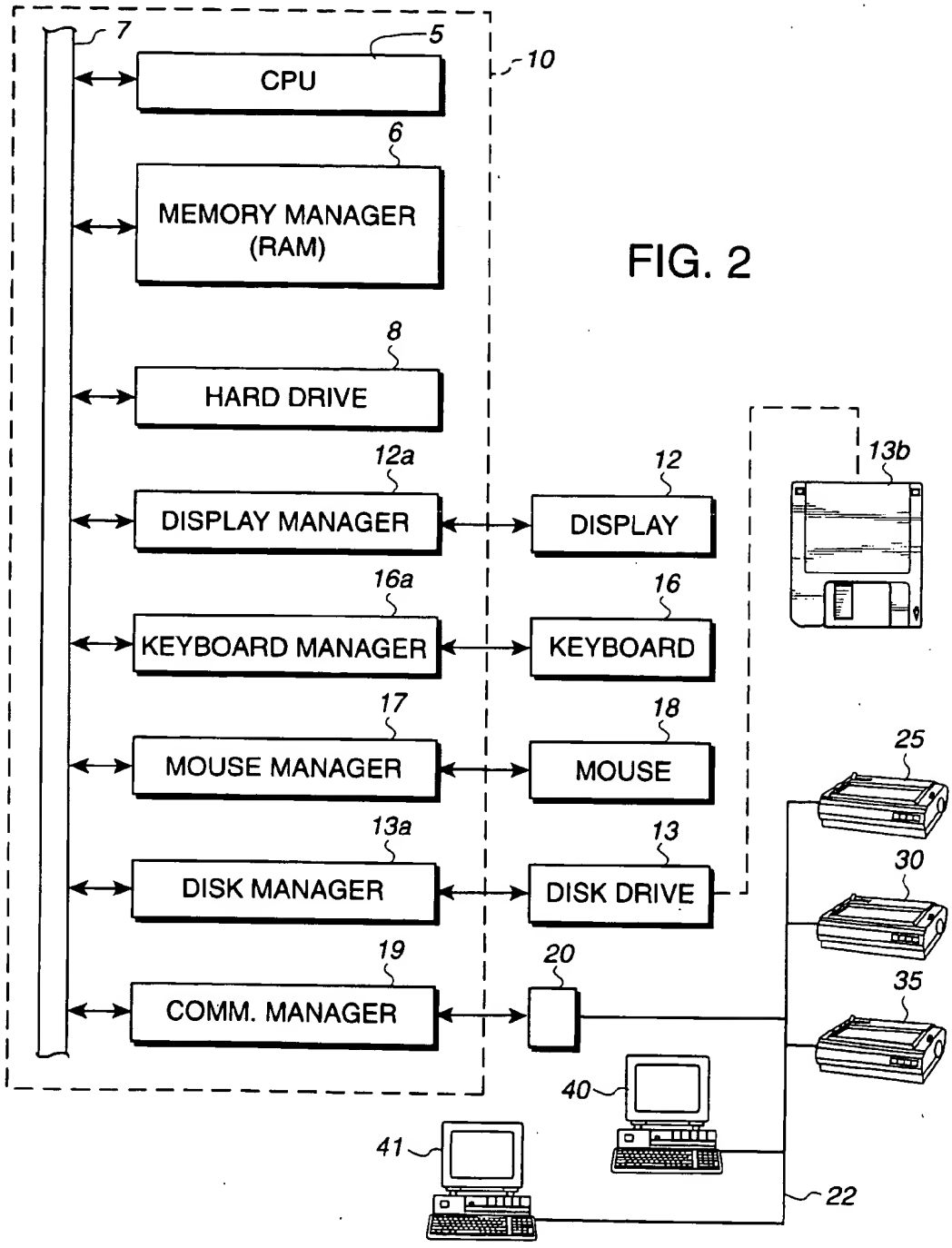

7
5
10
CPU
6
MEMORY MANAGER (RAM)
8
HARD DRIVE
12a
12
13b
DISPLAY MANAGER
DISPLAY
16a
16
KEYBOARD MANAGER
KEYBOARD
17
18
25
MOUSE MANAGER
MOUSE
13a
13
30
DISK MANAGER
DISK DRIVE
19
20
35
COMM. MANAGER
40
41
22

FIG. 2

FIG. 3

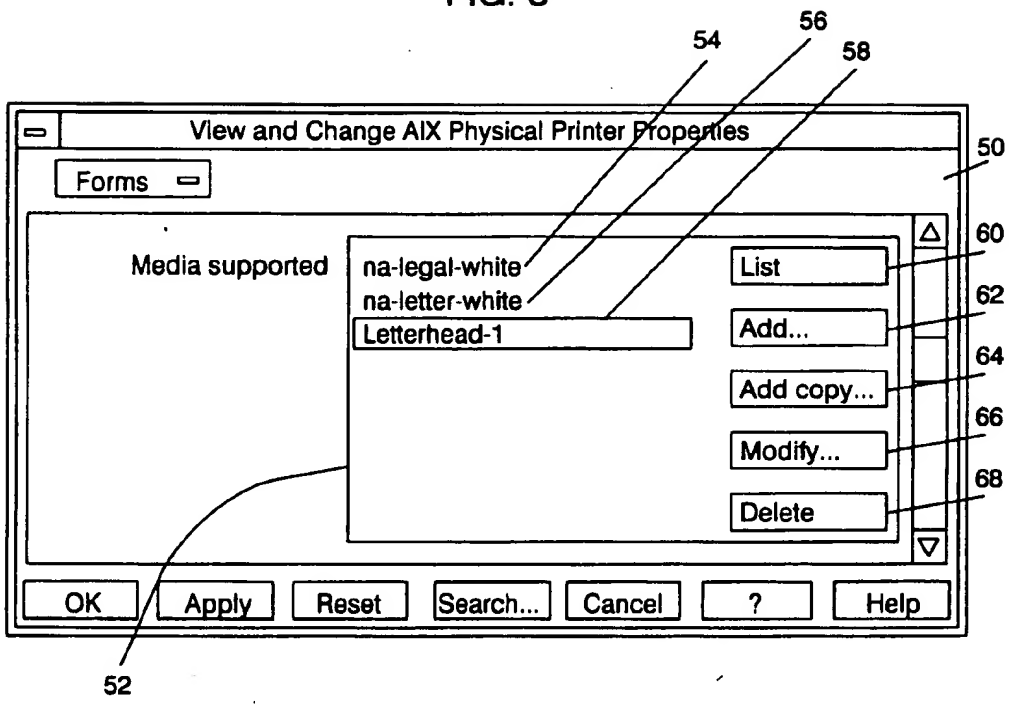

54
56
58
View and Change AIX Physical Printer Properties
50
Forms
Media supported
na-legal-white
na-letter-white
Letterhead-1
List
Add...
Add copy...
Modify...
Delete
60
62
64
66
68
OK
Apply
Reset
Search...
Cancel
?
Help
52

FIG. 4

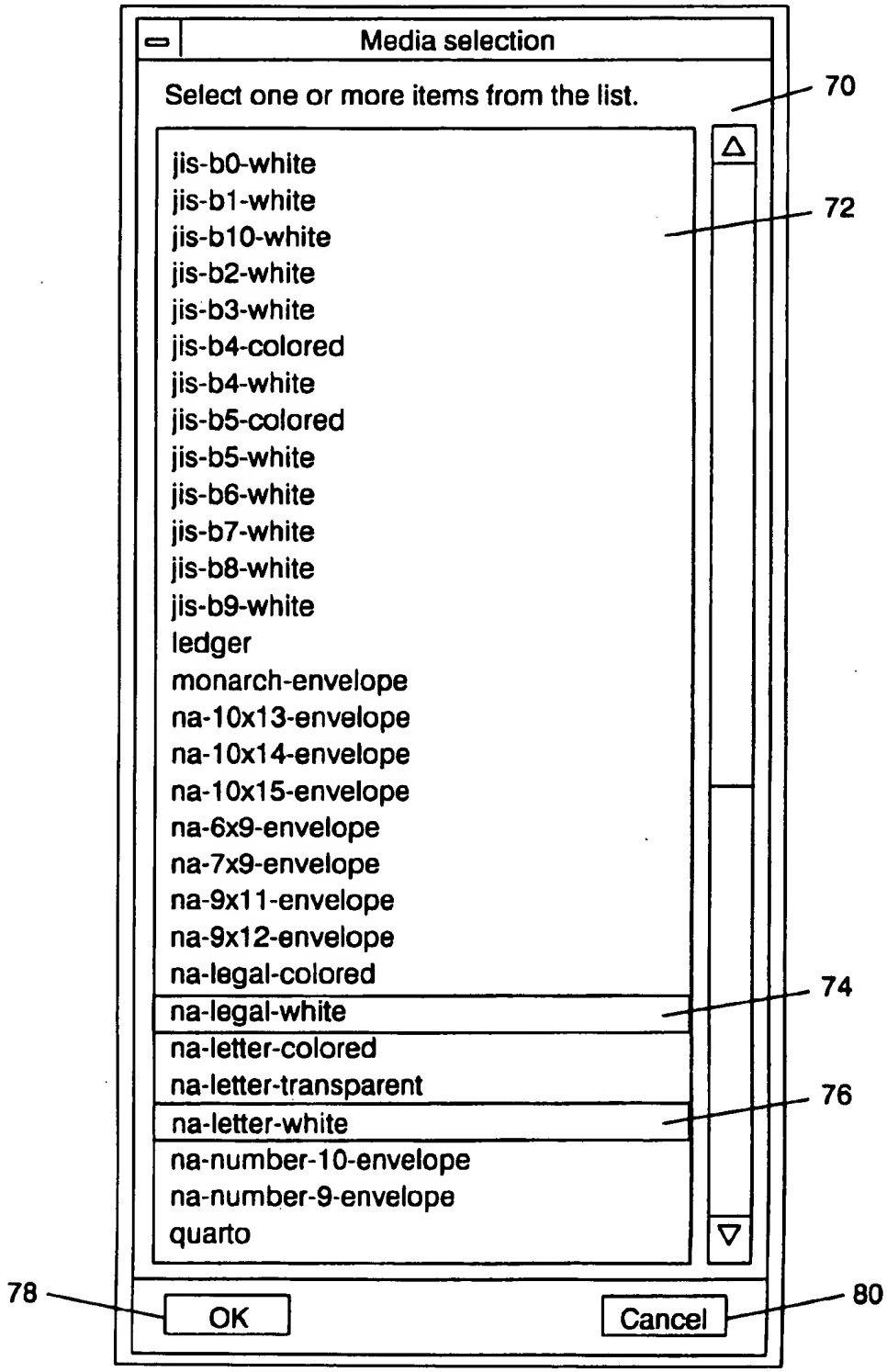

Media selection
Select one or more items from the list.
70
72
jis-b0-white
jis-b1-white
jis-b10-white
jis-b2-white
jis-b3-white
jis-b4-colored
jis-b4-white
jis-b5-colored
jis-b5-white
jis-b6-white
jis-b7-white
jis-b8-white
jis-b9-white
ledger
monarch-envelope
na-10x13-envelope
na-10x14-envelope
na-10x15-envelope
na-6x9-envelope
na-7x9-envelope
na-9x11-envelope
na-9x12-envelope
na-legal-colored
74
na-legal-white
na-letter-colored
na-letter-transparent
76
na-letter-white
na-number-10-envelope
na-number-9-envelope
quarto
78
OK
Cancel
80

FIG. 5

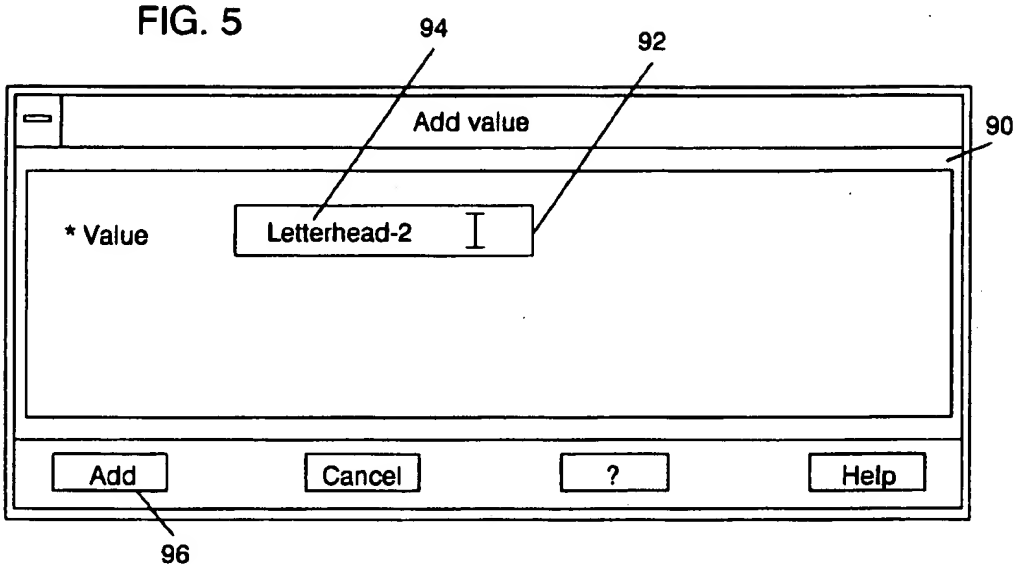

94
92
Add value
90
* Value
Letterhead-2
Add
Cancel
?
Help
96

FIG. 6

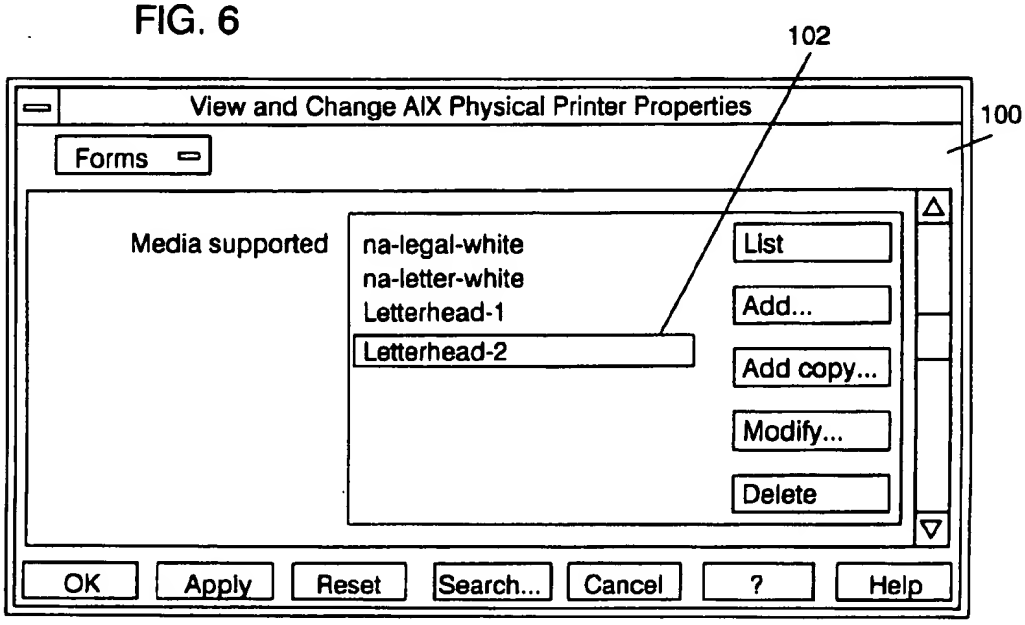

102
View and Change AIX Physical Printer Properties
100
Forms
Media supported
na-legal-white
na-letter-white
Letterhead-1
Letterhead-2
List
Add...
Add copy...
Modify...
Delete
OK
Apply
Reset
Search...
Cancel
?
Help

FIG. 7

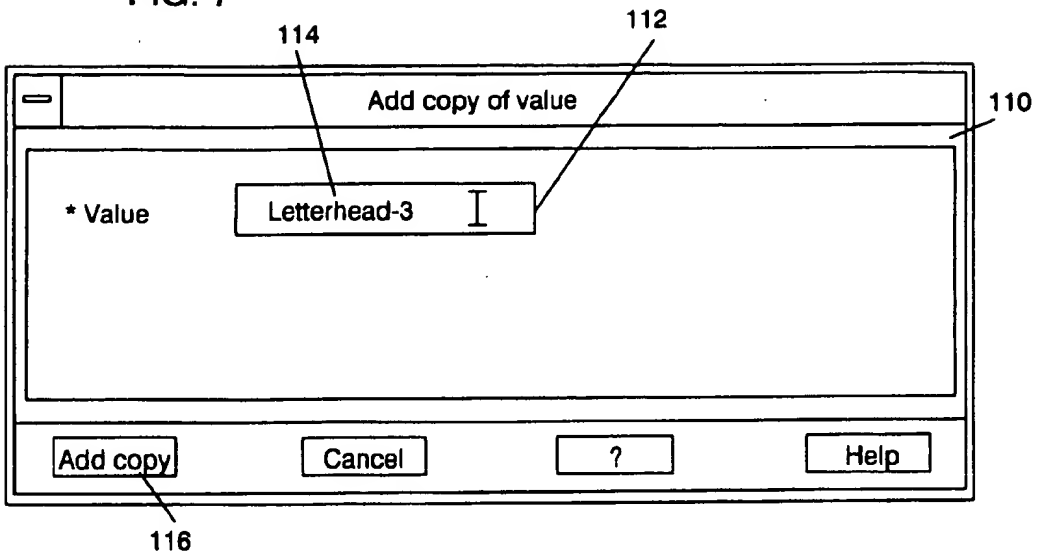

114
112
Add copy of value
110
* Value
Letterhead-3
Add copy
Cancel
?
Help
116

FIG. 8

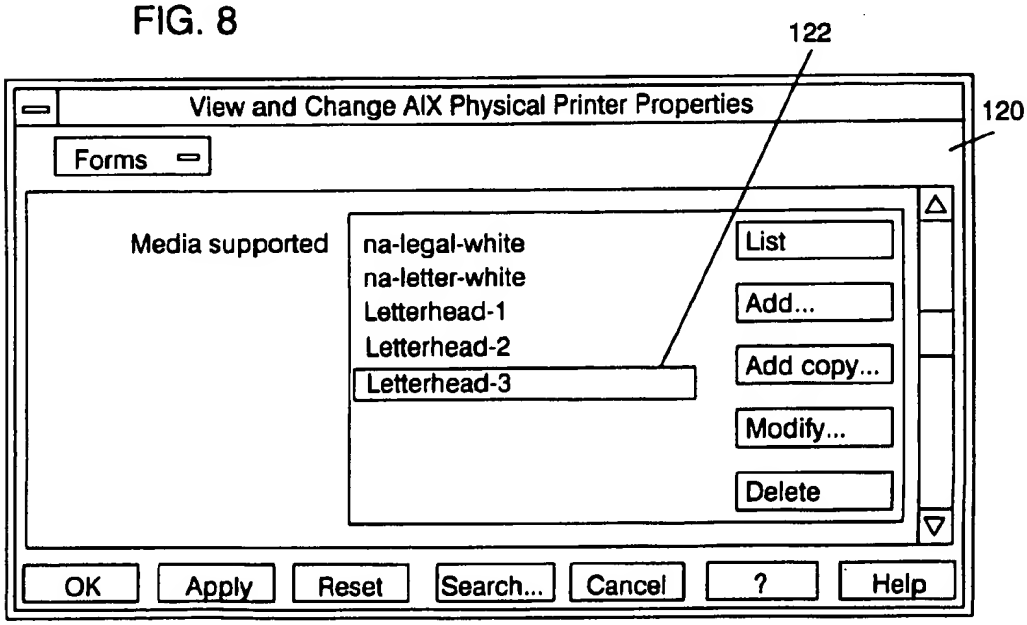

122
View and Change AIX Physical Printer Properties
120
Forms
Media supported
na-legal-white
na-letter-white
Letterhead-1
Letterhead-2
Letterhead-3
List
Add...
Add copy...
Modify...
Delete
OK
Apply
Reset
Search...
Cancel
?
Help

FIG. 9

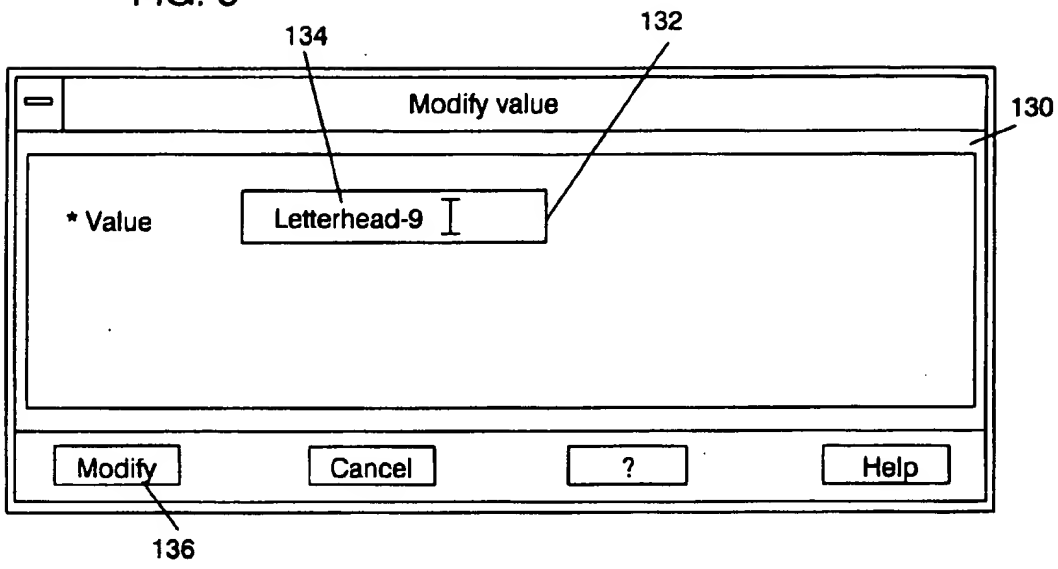

134
132
Modify value
130
* Value
Letterhead-9
Modify
Cancel
?
Help
136

FIG. 10

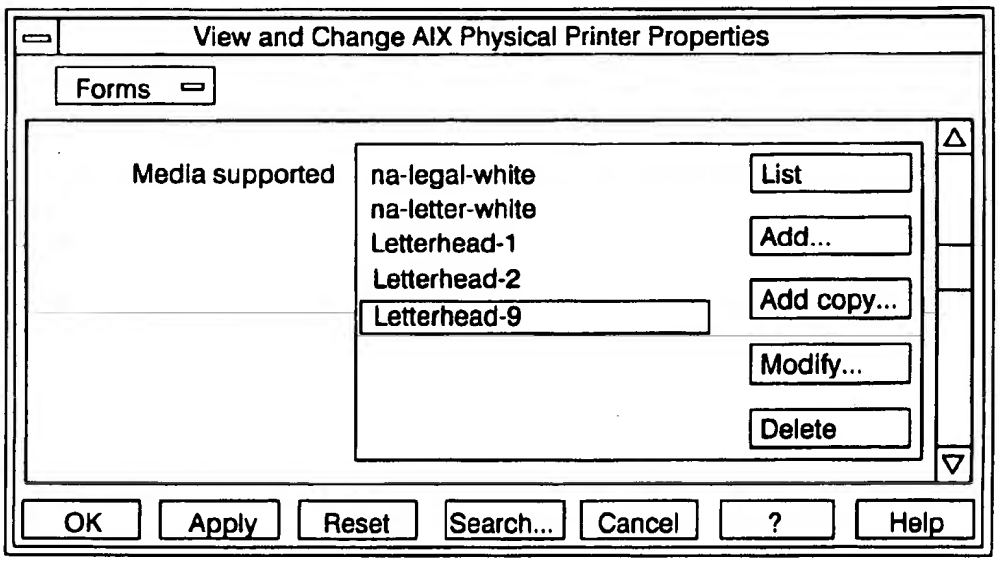

View and Change AIX Physical Printer Properties
Forms
Media supported
na-legal-white
na-letter-white
Letterhead-1
Letterhead-2
Letterhead-9
List
Add...
Add copy...
Modify...
Delete
OK
Apply
Reset
Search...
Cancel
?
Help

FIG. 11

FIG. 11A

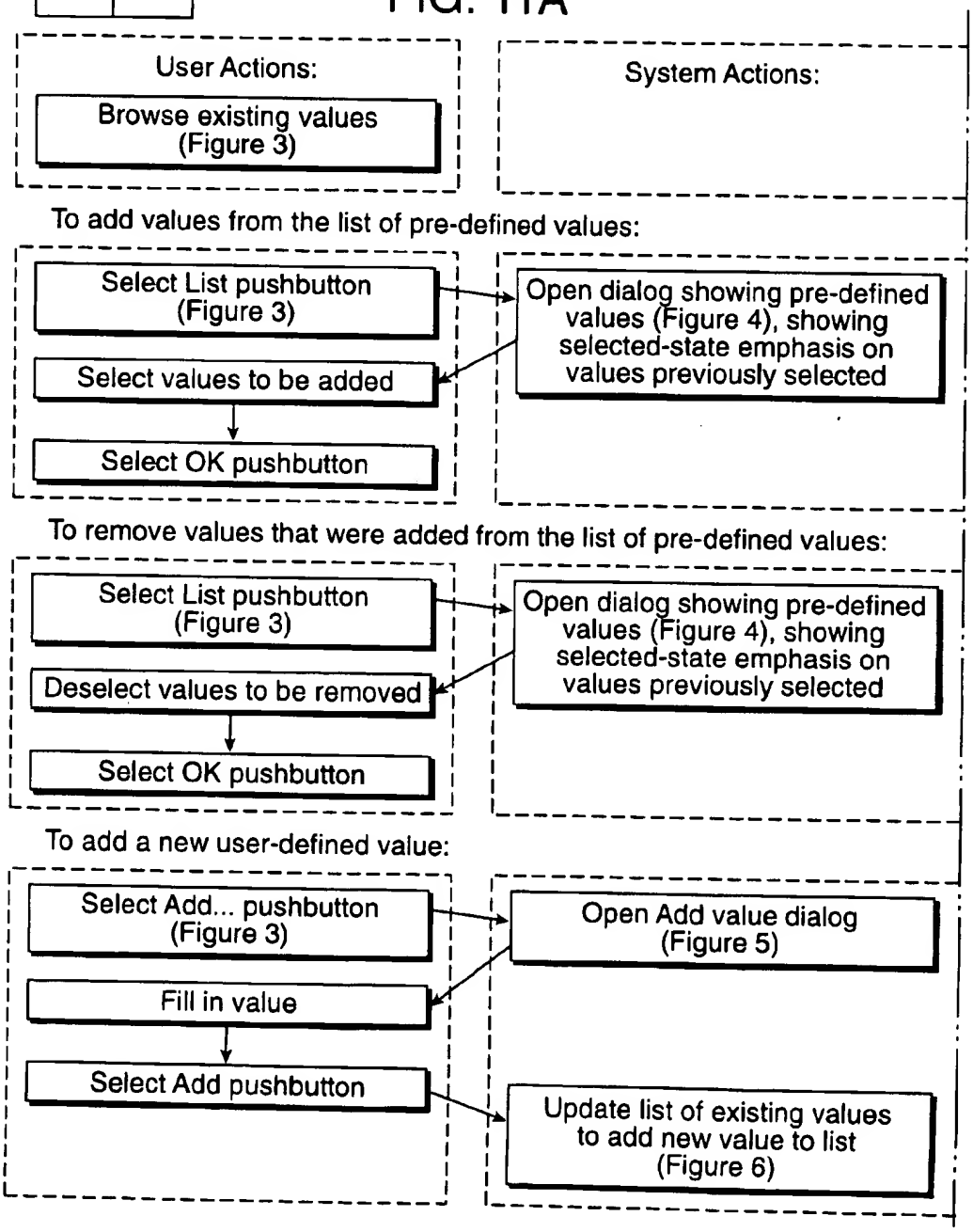
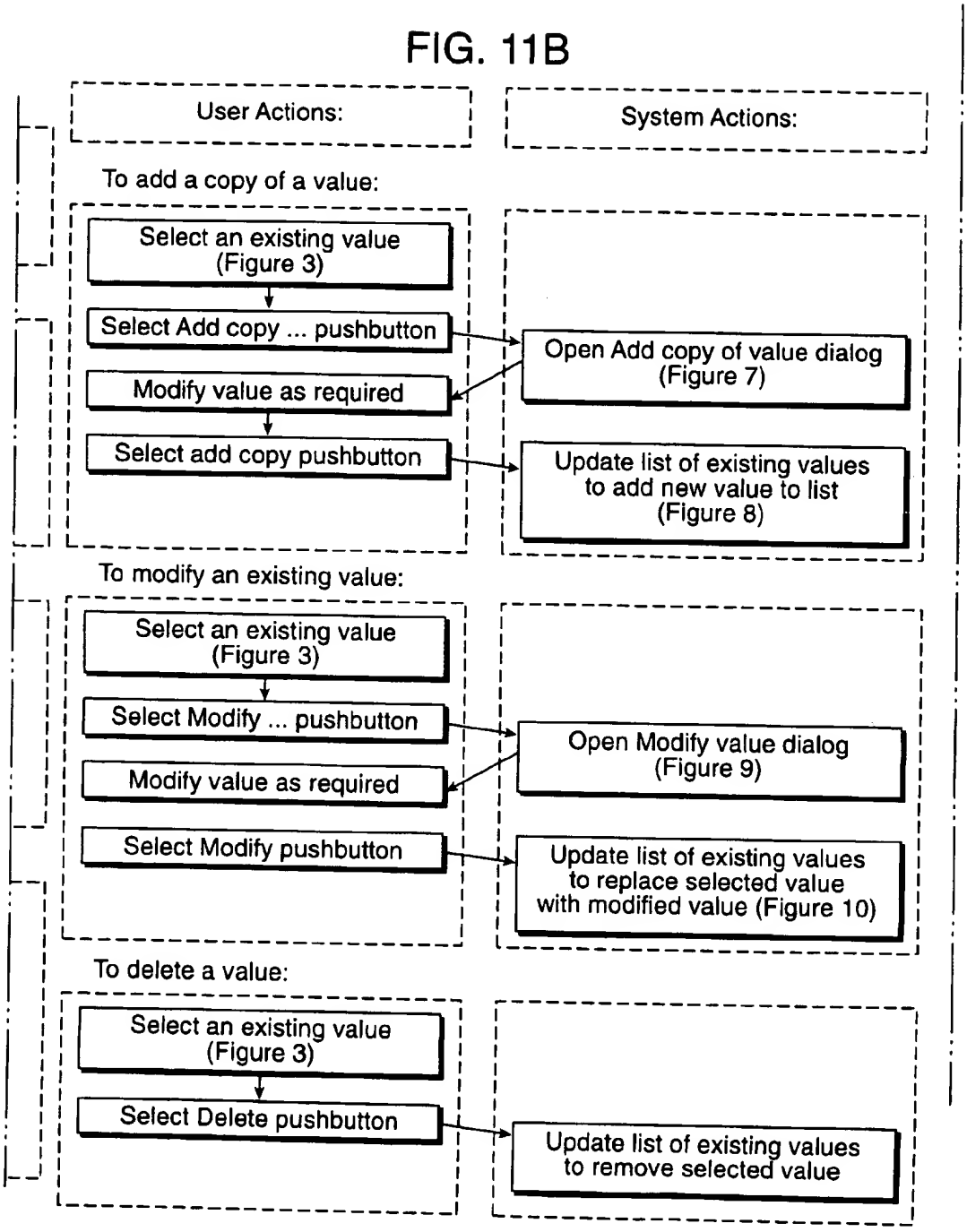


FIG. 11B


User Actions:
System Actions:
To add a copy of a value:
Select an existing value (Figure 3)
Select Add copy ... pushbutton
Modify value as required
Select add copy pushbutton
Open Add copy of value dialog (Figure 7)
Update list of existing values to add new value to list (Figure 8)
To modify an existing value:
Select an existing value (Figure 3)
Select Modify ... pushbutton
Modify value as required
Select Modify pushbutton
Open Modify value dialog (Figure 9)
Update list of existing values to replace selected value with modified value (Figure 10)
To delete a value:
Select an existing value (Figure 3)
Select Delete pushbutton
Update list of existing values to remove selected value

GRAPHICAL INTERFACE METHOD, APPARATUS AND APPLICATION FOR CREATING A LIST FROM PRE-DEFINED AND USER-DEFINED VALUES

This application is a Continuation of application Ser. No. 08/696,752, filed Aug. 14, 1996, which application(s) are incorporated herein by reference.

BACKGROUND OF THE INVENTION AND STATE OF THE PRIOR ART

1. Field of the Invention

The present invention relates to graphical user interfaces. More particularly, the invention relates to a method, apparatus and application for creating a list from pre-defined and user-defined values.

2. Description of Related Art

In order to better understand the terms utilized in this patent application, a brief background definition section will be presented so that the reader will have a common understanding of the terms employed and associated with the present invention.

A "user interface" is a group of techniques and mechanisms that a person employs to interact with an object. The user interface is developed to fit the needs or requirements of the users who use the object. Commonly known user interfaces can include telephone push buttons or dials, or pushbuttons such as on a VCR or a television set remote. With a computer, many interfaces not only allow the user to communicate with the computer but also allow the computer to communicate with the user. These would include (1) command-line user interfaces (i.e., user remembered commands which he/she enters, e.g. "C:>DIR" in which "DIR" is a typical DOS command entered at the "C" prompt); (2) menu-driven user interfaces which present an organized set of choices for the user, and (3) graphical user interfaces, ("GUI") in which the user points to and interacts with elements of the interface that are visible, for example by a "mouse" controlled arrow or cursor.

An example of a GUI user interface is that which is offered by International Business Machines Corporation (IBM) under the name "Common User Access" ("CUA"). This GUI incorporates elements of object orientation (i.e., the user's focus is on objects and the concept of applications is hidden). Object orientation of the interfaces allow for an interconnection of the working environment in which each element, called an "object," can interact with every other object. The objects users require to perform their tasks and the objects used by the operating environment can work cooperatively in one seamless interface. With object oriented programming using a GUI, the boundaries that distinguish applications from operating systems are no longer apparent or relevant to the user.

In connection with this patent application, an "object" means any visual component of a user interface that a user can work with as a unit, independent of other items, to perform a task. By way of example, a spreadsheet, one cell in a spreadsheet, a bar chart, one bar in a bar chart, a report, a paragraph in a report, a database, one record in a database, and a printer are all objects. Each object can be represented by one or more graphic images, called "icons," with which a user interacts, much as a user interacts with objects in the real world (NOTE: In the real world, an object might be an item that a person requires to perform work. As an example, an architect's objects might include a scale, T-square, and a sharp pencil, while an accountant's objects might include a ledger and a calculator.) However, it is not required that an object always be represented by an icon, and not all interaction is accomplished by way of icons.

While classification of objects may follow many different definitions, each class of objects has a primary purpose that separates it from the other classes. A class may be looked at as a group of objects that have similar behavior and information structures. In addition, each of the objects enumerated and defined below may contain other objects. There are three primary classes of objects. Each is discussed below.

(1) Container Object:

This object holds other objects. Its principal purpose is to provide the user with a way to hold or group related objects for easy access or retrieval. An operating system, e.g. OS/2® (a trademark of IBM Corporation) or Windows® (a trademark of Microsoft Corporation), typically provides a general-purpose container, for example a folder or a program group—that holds any type of object, including other containers. For example, imagine a program group (or folder) labeled "PRIVATE FOLDER—ICONS". In the program group are three folder icons labeled "REPORTS", "PORTFOLIO" and "LETTERS". By selecting with a mouse or other pointing device the icon "PORTFOLIO", another window may open showing three more icons labeled "OIL PAINTINGS", "WATERCOLORS", and "PORTRAITS". In turn, selecting any of those three icons may open additional windows with further icons representing further subdivisions, or cross-references (e. g., "CUSTOMERS").

(2) Data Objects:

The principal purpose of a data object is to convey information. This information may be textual or graphical information or even audio or video information. For example, a business report displayed on the computer monitor may contain textual information concerning sales of "gadgets" over the past few years (text object) to all customers and also may contain a bar chart (graphic object) to pictorially depict, on the same monitor screen, the sales information.

(3) Device Objects:

The principal purpose of a device object is to provide a communication vehicle between the computer and another physical or logical object. Many times the device object represents a physical object in the real world. For example, a mouse object or icon can represent the user's pointing device, and a modem object can represent the user's modem, or a printer object or icon can represent the user's printer. Other device objects are purely logical, e.g. an out-basket icon representing outgoing electronic mail; a wastebasket object or icon representing a way the user may "trash" or dispose of other objects.

As can be seen from the foregoing, a class of objects may be defined as a description of the common characteristics of several objects, or a template or model which represents how the objects contained in the class are structured. While there are further ways in which to define objects and classes of objects, typically each class of objects will include similar attributes, the values of which the user will alter, modify, replace or remove from time to time. For a more complete discussion of objects, attributes, object oriented interfaces etc. see "Object Oriented Interface Design: IBM Common User Access" (published by Que, ISBN 1-56529-170-0).

The present invention relates primarily to data objects. In a graphical user interface, the user frequently desires to construct a list of values. The values may be pre-defined or may be defined by the user. In addition to creating the list,

the user typically wants to modify the list, once created, by adding, deleting, or modifying values. Moreover, it is useful for the user to be able to copy values from one entry to another.

There are no standard solutions to these needs in the field of graphical user interfaces. Of course a variety of database products exist that provide various data management options. However, within graphical user interfaces there are no known solutions that adequately meet user needs.

SUMMARY OF THE INVENTION

In view of the above, it is a principal object of the present invention to provide a graphically oriented method, application and apparatus to construct and modify a list of values, whether the values are pre-defined or user-defined.

Another object of the present invention is to permit the user to add values to and delete values from a list by selecting and deselecting pre-defined values.

Another object of the present invention is to permit the user to add a user-defined value to a list.

Another object of the present invention is to allow the user to copy a value from a list, modify the value, and insert the modified value into the list while also retaining the original value in the list.

Yet another object of the present invention is to allow the user to copy a value from a list, modify the value, and replace the original value with the modified value in the list.

Still another object of the present invention is to allow the user to delete a value from a list.

The invention is carried out in the following environment. The computer system has at least a visual operator interface, an operating system for operating applications within the computer system, and memory for storing at least part, preferably all, of an application. The present invention provides a method, apparatus, and application for constructing in a graphical user interface a list of values, whether the values are pre-defined or user-defined. Also disclosed is a means for adding, modifying, and deleting values. Additionally disclosed is a means for copying selected values, modifying the selected values, and inserting the modified values into the list, while also retaining the original value in the list.

Other objects of the invention and a more complete understanding of the invention may be had by referring to the following description taken in conjunction with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWING(S)

FIG. **1** illustrates a typical desktop computer system which may be employed to practice the novel method and application of the present invention;

FIG. **2** is a block diagram illustrating a sample configuration of the computer system shown in FIG. **1**;

FIG. **3** is a typical window, in accordance with the present invention, showing a list of values in a user-list box;

FIG. **4** is a typical window showing how pre-defined values are selected for the list of values in the user-list box;

FIG. **5** is a typical window showing how user-defined values are added to the list of values in the user-list box;

FIG. **6** is a typical window showing the added user defined value in the user-list box.

FIG. **7** is a typical window showing the copying of a value already in the user-list box, and then modification of the copy;

FIG. **8** is a typical window showing the added copy, as modified, in the user-list box;

FIG. **9** is a typical window showing the modification of an existing value in the user-list box;

FIG. **10** is a typical window showing the modified value in the user-list box; and

FIG. **11** is a flow chart illustrating the method of the present invention, including FIG. **11**A and FIG. **11**B.

DESCRIPTION OF THE ILLUSTRATIVE EMBODIMENT(S)

Turning now to the drawings, and especially FIGS. **1** and **2**, FIG. **1** diagrammatically shows a computer system **1** which may be connected to a Local Area Network system (LAN **20**) as shown in FIG. **2**.

As shown in FIG. **1**, the computer system **1** comprises a main chassis **10**, a display means or monitor **12**, a connected keyboard **16** and a pointing device, in the present instance a mouse **18** which is operator controlled to move a pointer cursor **12***b* (shown in FIG. **3**) on the display or monitor screen **12***c*. As shown in FIG. **2**, the chassis **10** includes a central processing unit, or "CPU" **5**, a memory manager and associated random access memory, or "RAM" **6**, a fixed disk or hard drive **8** (which may include its associated disk controller), a display manager **12***a* which is connected externally to the chassis **10** of the display **12**; a keyboard manager **16***a*, which through flexible cable (not shown) is connected to the keyboard **16**; a mouse manager **17** (which in some instances may form part of the display manager **12***a*, and may be in the form of a software driver) for reading the motion of the mouse **18** and its control mouse buttons (MB) **18***a* and **18***b*, shown in FIG. **1**. A disk manager or controller **13***a* which controls the action of the disk drive **13** (and an optional drive such as a magneto-optical or CD ROM drive **14**) shown in FIG. **1**, rounds out most of the major elements of the computer system **1**.

The pointer element or cursor **12***b* can be moved over the display screen **12***c* by movement of the mouse **18**. The mouse buttons (MB) **18***a* and **18***b* give commands to the operating system, usually through a software mouse driver provided by the mouse manufacturer. With the first mouse button (MB) **18***a* the operator can select an element indicated on the display screen **12***c* using the pointer or cursor **12***b*, i.e., signify that an action subsequently to be performed is to be carried out on the data represented by the indicated element on the display screen **12***c*. The system normally gives some visual feedback to the operator to indicate the element selected, such as a change in color, or a blocking of the icon. The second mouse button (MB) **18***b* may be a menu button, if desired. Conventionally, when the operator presses button **18***b*, a selection menu or dialog with system commands will appear on the display screen **12***c*. The operator may select an icon or item from the selection menu or input information into the dialog box as appropriate using the cursor **12***b* and the first mouse button (MB) **18***a*. Some menu items, if selected, may call up another menu or submenu for the operator to continue the selection process.

The use of a mouse and selection menus is well known in the art, for example U.S. Pat. No. 4,464,652 to Lapson et al. describes a selection menu of the pull-down type in combination with a mouse. It should be recognized, of course, that other cursor pointing devices may be employed, for example a joystick, ball and socket, or cursor keys on the keyboard.

The foregoing devices (and software drivers therefore) within the chassis **10** communicate with one another via a

bus 7. To round out the computer system 1, an operating system (not shown) must be employed. If the computer system is a typical IBM-based system, the operating system may be DOS-based and include a GUI interface such as contained in OS/2®, or WINDOWS®, or other operating system of choice. If the computer system is based upon RISC (reduced instruction set computer) architecture, then the operating system employed may be, in the instance of an IBM-based RISC architectured System/6000®, AIX. Alternatively, if the computer system 1 is a large host computer, such as a an IBM 3090, it may be running an operating system such as MVS or VM.

In the illustrated instance, the computer system 1 includes an I/O (Input/Output) manager or communications manager 19 (shown in FIG. 2) which serves to link the computer system for communications with the outside world such as to a systems printer, a modem or a LAN controller (such as a Token ring or ETHERNET or even through a modem employing SDLC) such as showm at 20 in FIG. 2. The LAN controller may be incorporated inside the computer system 1 or located externally as shown diagrammatically in FIG. 2, as desired. The LAN controller 20 may connect to other computer systems 40 and 41 as well as to other printers such as printers 25, 30 and 35 by communications cable 22 and the like. However the method and application of the present invention works equally well with multiple objects serviced by a single computer system.

Referring now to FIG. 3, the display screen 12c of the monitor 12 is shown with a window 50. As illustrated, a user-list box 52 contains three values, 54, 56 and 58. Value 54 is one value in the user-list box 52.

The values Shown in FIG. 3, are na-legal-white 54, na-letter-white 56, and Letterhead-1 58. In this example, na-legal-white 54 and na-letter-white 56 are pre-defined values, while Letterhead-1 58 is a user-defined value. Down the right hand side of window 50 are five pushbuttons 60, 62, 64, 66 and 68. The List Pushbutton 60 is selected to display a list of pre-defined values which can then be selected and/or deselected. The Add Pushbutton 62 is selected to add a user-defined value to the user-list box 52. The Add Copy Pushbutton 64 is selected to copy a value from the user-list box 52 to create another entry in the user-list box 52. The Modify Pushbutton. 66 is selected to modify an existing value in the user-list box 52. The Delete Pushbutton 68 is selected to delete an existing value in the user-list box 52. The operation of each of these pushbuttons is described in further detail below.

FIG. 4 shows a typical window 70 to demonstrate how pre-defined values are added to and deleted from the user-list box 52. When the user selects the List Pushbutton 60 in the window 50 in FIG. 3, the window 70 appears. The window 70 lists pre-defined values 72 which may be selected and deselected so as to add and delete the pre-defined values from the user-list box 52. Note that in this example there are two pre-defined values, na-legal-white 74 and na-letter-white 76, which are highlighted in the pre-defined value list 72. The highlighting indicates that these pre-defined values are currently in the user-defined list box 52 in FIG. 3. The user may select additional pre-defined values to be added to the user-list box 52, or may deselect values currently in the user-list box 52, which will remove these pre-defined values from the user-list box 52. When the user is done selecting and deselecting pre-defined values, the user selects the OK Pushbutton 78. The Cancel Pushbutton 80 may also be used to return to the user-list box 52 without making any changes.

FIG. 5 shows a typical window 90 to demonstrate how user-defined values are added to the user-list box 52. When the user selects the Add Pushbutton 62 in the window 50 in FIG. 3, the window 90 appears. The window 90 contains a Value Box 92. To add a user-defined value to the user-list box 52, the user enters the value to be added. In this example, the user-defined value to be added is Letterhead-2 94. The user then selects the Add Pushbutton 96 at the bottom of window 90. The window 90 then closes and, as shown in FIG. 6, the window 100 appears. Window 100 is identical to window 50, except. that the user-defined value 102 added in window 90 is shown in the user-list box 52.

Assume now that the user desires to copy value 102 to create a fifth entry in user-list box 52. The user selects Add Copy Pushbutton 64. This causes window 110 to appear, as shown in FIG. 7. The value 102, from FIG. 6, is "pre-filled in" to the Value Box 112 on window 110. If the user then desires to make certain modifications to the value 102, this can be done without the user having to re-enter all of the information for the value 102.

In this example, the user changed the value Letterhead-2 to Letterhead-3 114. When the modifications are complete, the user selects the Add Copy Pushbutton 116. This causes window 120 to appear, as shown in FIG. 8. As shown in FIG. 8, the new value 122 has been added to the user-list box 52. Note that both the original value and the modified value now appear in the user-list box.

Now suppose the user desires to modify value 122. The user selects the Modify Pushbutton 66. This causes the window 130 shown in FIG. 9 to appear. Window 130 contains a Value Box 132 which is "pre-filled-in" with the information for value 122 showing in the user-list box 52 and as previously entered by the user. The user can make minor changes to the user-defined value 134 as shown in FIG. 9. In this example, the user changed Letterhead-3 to Letterhead-9 134. Of course, other modifications are also possible. The modifications shown are examples only.

When the modifications are complete, the user selects the Modify Pushbutton 136 as shown in FIG. 9. FIG. 10 shows user-list box 52 with value 134 modified as shown in FIG. 9. Note that when modifying a user-defined value, only the modified value appears in the user-list box.

Although not illustrated in the Figures, the user also can delete a value in the user-list box 52. To delete a value, the Delete Pushbutton 68 is selected.

FIG. 11 shows the method of the present invention.

Although the invention has been described with a certain degree of particularity, it should be recognized that elements thereof may be altered by person(s) skilled in the art without departing from the spirit and scope of the invention. The invention is limited only by the following claims and their equivalents.

What is claimed is:

1. A method for creating a window displaying user defined printer values in a graphical user interface (GUI) displayed on a computer monitor attached to a computer, wherein an input device in communication with the computer is used to control display operations in the GUI, wherein a printer is in communication with the computer, comprising the steps of:

displaying a first window of pre-defined printer values;

selecting, with the input device, at least one pre-defined printer value displayed in the first window;

displaying in a second window the selected pre-defined printer value;

modifying, with the input device, the pre-defined printer value displayed in the second window;

displaying the modified pre-defined printer value in the second window; and

processing with the computer the modified pre-defined printer value to control printer operations.

2. The method of claim 1, wherein the step of modifying the pre-defined printer value comprises changing the displayed name of the modified pre-defined printer value from a first name to a second name, and wherein the step of displaying the modified pre-defined printer value in the second window comprises displaying the second name of the modified pre-defined printer value.

3. The method of claim 2, wherein the step of processing the modified pre-defined printer value having the second name generates the same printer operation outcome as processing the pre-defined printer value having the first name.

4. The method of claim 1, wherein the step of modifying the pre-defined printer value comprises creating a copy of the predefined printer value, wherein the step of displaying the modified pre-defined printer value in the second window comprises displaying both the copy of the pre-defined printer value and the pre-defined printer value in the second window.

5. The method of claim 4, further comprising the step of changing a displayed name of the copy of the predefined printer value from a first name to a second name, wherein the step of displaying both the copy of the pre-defined printer value and the pre-defined printer value comprises displaying the pre-defined printer value with the first name and displaying the copy of the pre-defined printer value with the second name.

6. A system for modifying user defined printer values, comprising:

a computer;

an input device in communication with the computer for entering data into the computer;

a display screen in communication with the computer, wherein the display screen displays a graphical user interface (GUI), and wherein the input device is capable of controlling display operations in the GUI;

a printer in communication with the computer;

program logic executed by the computer, including:

(i) means for displaying a first window of pre-defined printer values;

(ii) means for accepting a selection, with the input device, of at least one pre-defined printer value displayed in the first window;

(iii) means for displaying in a second window the selected pre-defined printer values;

(iv) means for modifying the pre-defined printer value displayed in the second window;

(v) means for displaying the modified pre-defined printer value in the second window; and

(vi) means for processing the modified pre-defined printer value to control printer operations.

7. The system of claim 6, wherein the program logic further comprises:

means for changing the displayed name of the pre-defined printer value from a first name to a second name; and

means for displaying the second name of the pre-defined printer value.

8. The system of claim 7, wherein the means for processing the modified pre-defined printer value having the second name generates the same printer operation outcome as processing the pre-defined printer value having the first name.

9. The system of claim 6, wherein the program logic further comprises:

means for creating a copy of the predefined printer value; and

means for displaying both the copy of the pre-defined printer value and the pre-defined printer value in the second window.

10. The system of claim 9, wherein the program logic further includes:

means for changing a displayed name of the copy of the predefined printer value from a first name to a second name; and

means for displaying the pre-defined printer value with the first name and displaying the copy of the pre-defined printer value with the second name in the second window.

11. An article of manufacture for use in programming a computer to allow a user to control the display of printer values in a graphical user interface (GUI) on a display monitor controlled by the computer, the article of manufacture comprising at least one computer readable storage device including at least one computer program embedded therein that causes the computer to perform the steps of:

displaying a first window of pre-defined printer values;

indicating the selection, in response to data entered by an input device in communication with the computer, of at least one pre-defined printer value displayed in the first window;

displaying in a second window the selected pre-defined printer value;

modifying, in response to data entered by the input device, the pre-defined printer value displayed in the second window;

displaying the modified pre-defined printer value in the second window; and

processing the modified pre-defined printer value to control the operation of the printer.

12. The article of manufacture of claim 11, wherein the step of modifying the pre-defined printer value comprises causing the computer, in response to data entered by the input device, to change the displayed name of the modified pre-defined printer value from a first name to a second name, and wherein the step of displaying the modified pre-defined printer value in the second window comprises the computer displaying the second name of the modified pre-defined printer value.

13. The article of manufacture of claim 11, wherein the step of the computer processing the modified pre-defined printer value having the second name comprises the computer generating the same printer operation outcome as processing the pre-defined printer value having the first name would cause the computer to generate.

14. The article of manufacture of claim 11, wherein the step of modifying the predefined printer value comprises causing the computer, in response to data entered by the input device, to create a copy of the predefined printer value, wherein the step of displaying the modified pre-defined printer value in the second window comprises displaying both the copy of the pre-defined printer value and the pre-defined printer value in the second window.

15. The article of manufacture of claim 14, further comprises causing the computer, in response to data entered by the input device, to change a displayed name of the copy of the predefined printer value from a first name to a second name, wherein the step of displaying both the copy of the pre-defined printer value and the pre-defined printer value comprises displaying the pre-defined printer value with the first name abd displaying the copy of the pre-defined printer value with the second name.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 5,982,367
DATED : November 9, 1999
INVENTOR(S) : Alimpich et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Title page, Item [54] and Column 1, lines 1-4,
Delete the Title, and insert -- **METHOD AND SYSTEM FOR CREATING A LIST FROM PRE-DEFINED AND USER-DEFINED VALUES** --

Column 2,
Line 22, before "By", delete "".

Column 5,
Line 19, delete "showm" and insert -- shown --
Line 32, delete "Shown" and insert -- shown --
Line 43, after "Pushbutton", delete "."

Column 6,
Line 9, after "except", delete "."

Column 8,
Line 64, delete "abd" and insert -- and --

Signed and Sealed this

Twenty-first Day of October, 2003

JAMES E. ROGAN
Director of the United States Patent and Trademark Office

US005818444A

United States Patent [19]

Alimpich et al.

[11] Patent Number: 5,818,444

[45] Date of Patent: Oct. 6, 1998

[54] **METHOD, APPARATUS AND APPLICATION FOR OBJECT SELECTIVE BUT GLOBAL ATTRIBUTE MODIFICATION**

[75] Inventors: **Claudia C. Alimpich**, Boulder; **Gerald D. Boldt**; **Calvin Larry Doescher**, both of Longmont; **Joan Stagaman Goddard**, Boulder; **Luana L. Vigil**, Longmount, all of Colo.; **Minh Trong Vo**, Mountain View, Calif.; **James Philip Wittig**, Boulder, Colo.

[73] Assignee: **International Business Machines Corporation**, Armonk, N.Y.

[21] Appl. No.: **696,762**

[22] Filed: **Aug. 14, 1996**

[51] **Int. Cl.**[6] **G06F 15/00**

[52] **U.S. Cl.** **345/333**

[58] **Field of Search** 345/333, 349, 345/350, 356, 357, 433

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,464,652	8/1984	Lapson et al.	340/710
5,001,654	3/1991	Winiger et al.	364/523

(List continued on next page.)

FOREIGN PATENT DOCUMENTS

2097540	12/1994	Canada .
0 587 394 A1	3/1994	European Pat. Off. .
0 622 728 A1	11/1994	European Pat. Off. .
4-361373	12/1992	Japan .
5-313845	11/1993	Japan .
6-4117	1/1994	Japan .
6-215095	8/1994	Japan .
7-129597	5/1995	Japan .

OTHER PUBLICATIONS

"Device Independent Graphics Using Dynamic Generic Operator Selection," *IBM Technical Disclosure Bulletin*, Apr. 1983, vol. 25, No. 11A, pp. 5477–5480.

"Error–Tolerant Dynamic Allocation of Command Processing Work Space," *IBM Technical Disclosure Bulletin*, Jun. 1984, vol. 27, No. 1B, pp. 584–586.

"Means for Computing the Max of a Set of Variables Distributed Across Many Processors," *IBM Technical Disclosure Bulletin*, Sep. 1990, vol. 33, No. 4, pp. 8–12.

"Graphical User Interface for the Distributed System Namespace," *IBM Technical Disclosure Bulletin*, Jul. 1992, vol. 35, No. 2, pp. 335–336.

"Graphical Query System," *IBM Technical Disclosure Bulletin*, Nov. 1993, vol. 36, No. 11, pp. 615–616.

"Configuration Data Set Build Batch Program," *IBM Technical Disclosure Bulletin*, Nov. 1993, vol. 36, No. 11, p. 571.

Self–Contained Reusable Programmed Components, *IBM Technical Disclosure Bulletin*, Jul. 1995, vol. 38, No. 7, pp. 283–285.

"IBM Printing Systems Manager for AIX Overview," *International Business Machines Corporation*, Second Edition, Feb. 1996.

(List continued on next page.)

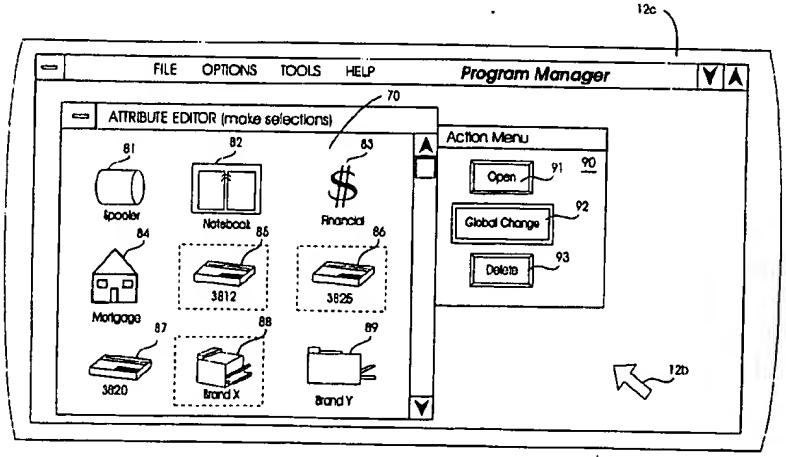
Primary Examiner—Phu K. Nguyen

Attorney, Agent, or Firm—Jenkens & Gilchrist

[57] **ABSTRACT**

An application, method and apparatus for revision of attributes of selected objects as controlled by a computer system. The computer system has at least a visual operator interface, an operating system for operating applications within the computer system, and memory for storing the application therein. Selected objects, such as other applications, parts of applications such as cells in a spread sheet, or real objects such as printers, are designated for selected attribute revision. A global change operation for all of the designated objects having properties to be revised is selected. The attributes are revised as desired and the revised attributes are propagated to the designated objects. Also disclosed is a method, apparatus and application for indicating the desired revisions to the attributes using a model object. The model object method, apparatus and application is an alternative embodiment of the invention.

6 Claims, 10 Drawing Sheets



U.S. PATENT DOCUMENTS

5,062,060	10/1991	Kolnick	364/521
5,072,412	12/1991	Henderson	395/159
5,095,512	3/1992	Roberts et al.	382/56
5,117,372	5/1992	Petty	395/161
5,119,476	6/1992	Texier	395/157
5,121,477	6/1992	Koopmans et al.	395/156
5,140,677	8/1992	Fleming et al.	395/159
5,140,678	8/1992	Torres	395/159
5,164,911	11/1992	Juran et al.	364/578
5,206,950	4/1993	Geary et al.	395/600
5,208,907	5/1993	Shelton et al.	395/149
5,228,123	7/1993	Heckel	395/155
5,233,687	8/1993	Henderson et al.	395/158
5,247,651	9/1993	Clarisse et al.	395/500
5,249,265	9/1993	Liang	395/160
5,255,359	10/1993	Ebbers et al.	395/161
5,276,901	1/1994	Howell et al.	395/800
5,287,447	2/1994	Miller et al.	395/157
5,307,451	4/1994	Clark	395/127
5,315,703	5/1994	Matheny et al.	395/164
5,317,687	5/1994	Torres	395/159
5,317,730	5/1994	Moore et al.	395/600
5,367,619	11/1994	Dipaolo et al.	395/149
5,371,844	12/1994	Andrew et al.	395/155
5,377,317	12/1994	Bates et al.	395/157
5,388,255	2/1995	Pytlik et al.	395/600
5,394,521	2/1995	Henderson et al.	395/158
5,404,439	4/1995	Moran et al.	395/155
5,410,695	4/1995	Frey et al.	395/650
5,410,704	4/1995	Norden-Paul et al.	395/700
5,412,776	5/1995	Bloomfield et al.	395/160
5,414,806	5/1995	Richards	395/135
5,416,900	5/1995	Blanchard et al.	395/155
5,418,950	5/1995	Li et al.	395/600
5,428,554	6/1995	Laskoski	364/550
5,428,776	6/1995	Rothfield	395/600
5,438,659	8/1995	Notess et al.	395/155
5,450,545	9/1995	Martin et al.	395/700
5,454,071	9/1995	Siverbrook et al.	395/141
5,454,106	9/1995	Burns et al.	395/600
5,459,825	10/1995	Anderson et al.	395/133
5,459,832	10/1995	Wolf et al.	395/155
5,463,724	10/1995	Anderson et al.	395/148
5,473,745	12/1995	Berry et al.	395/157
5,479,599	12/1995	Rockwell et al.	395/155
5,481,666	1/1996	Nguyen et al.	395/159
5,483,651	1/1996	Adams et al.	395/600
5,487,141	1/1996	Cain et al.	395/135
5,491,795	2/1996	Beaudet et al.	395/159
5,497,454	3/1996	Bates et al.	395/158
5,497,484	3/1996	Potter et al.	395/600
5,678,014	10/1997	Malamud et al.	345/333

OTHER PUBLICATIONS

"IBM Printing Systems Manager for AIX Administrating," *International Business Machines Corporation*, 1995.

"Matching Three–Dimensional Objects Using a Relational Paradigm," *Pattern Recognition*, vol. 17, No. 4, pp. 385–405, 1984.

"A Multicolumn List–Box Container for OS/2," *Dr. Dobb's Journal*, May 1994, vol. 19, No. 5, pp. 90–94.

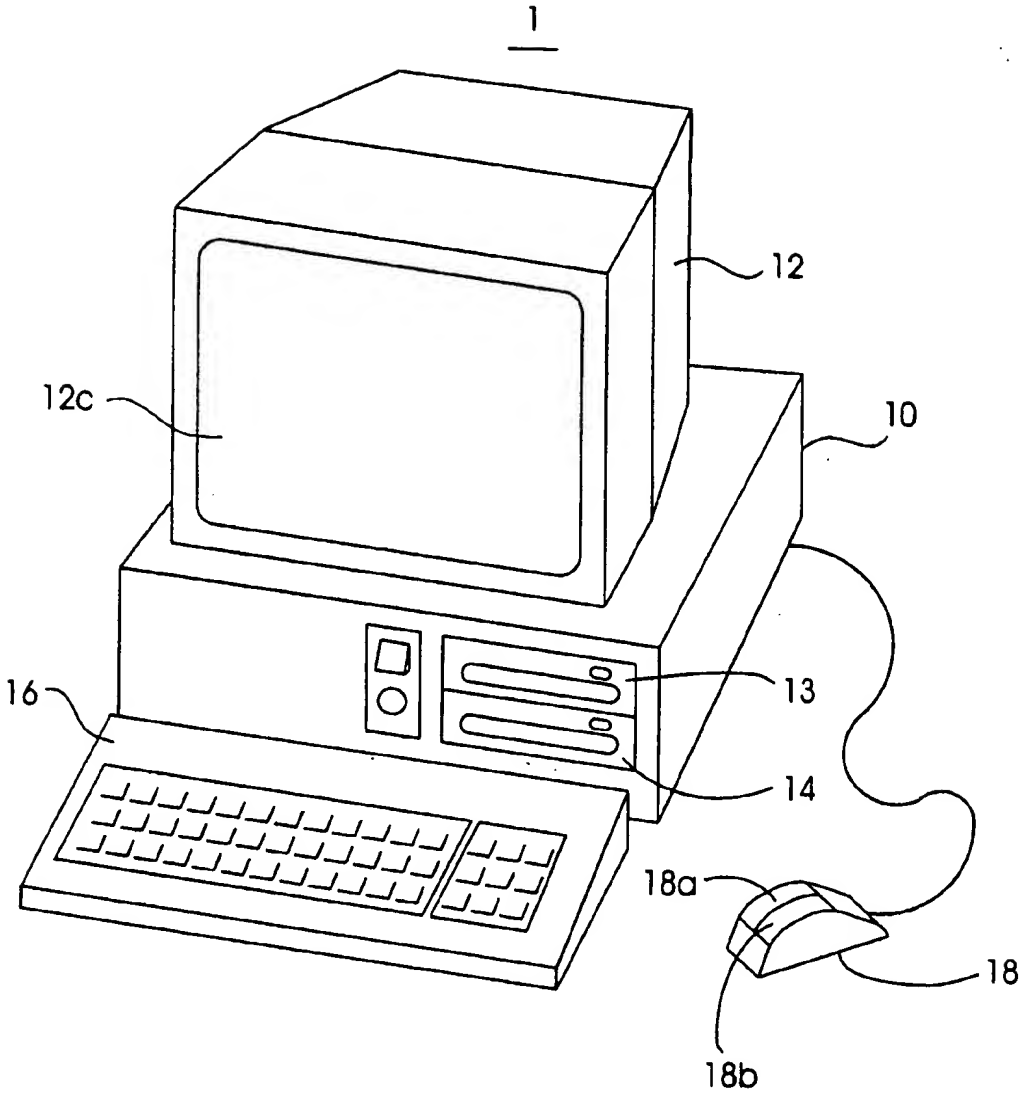


FIG. 1

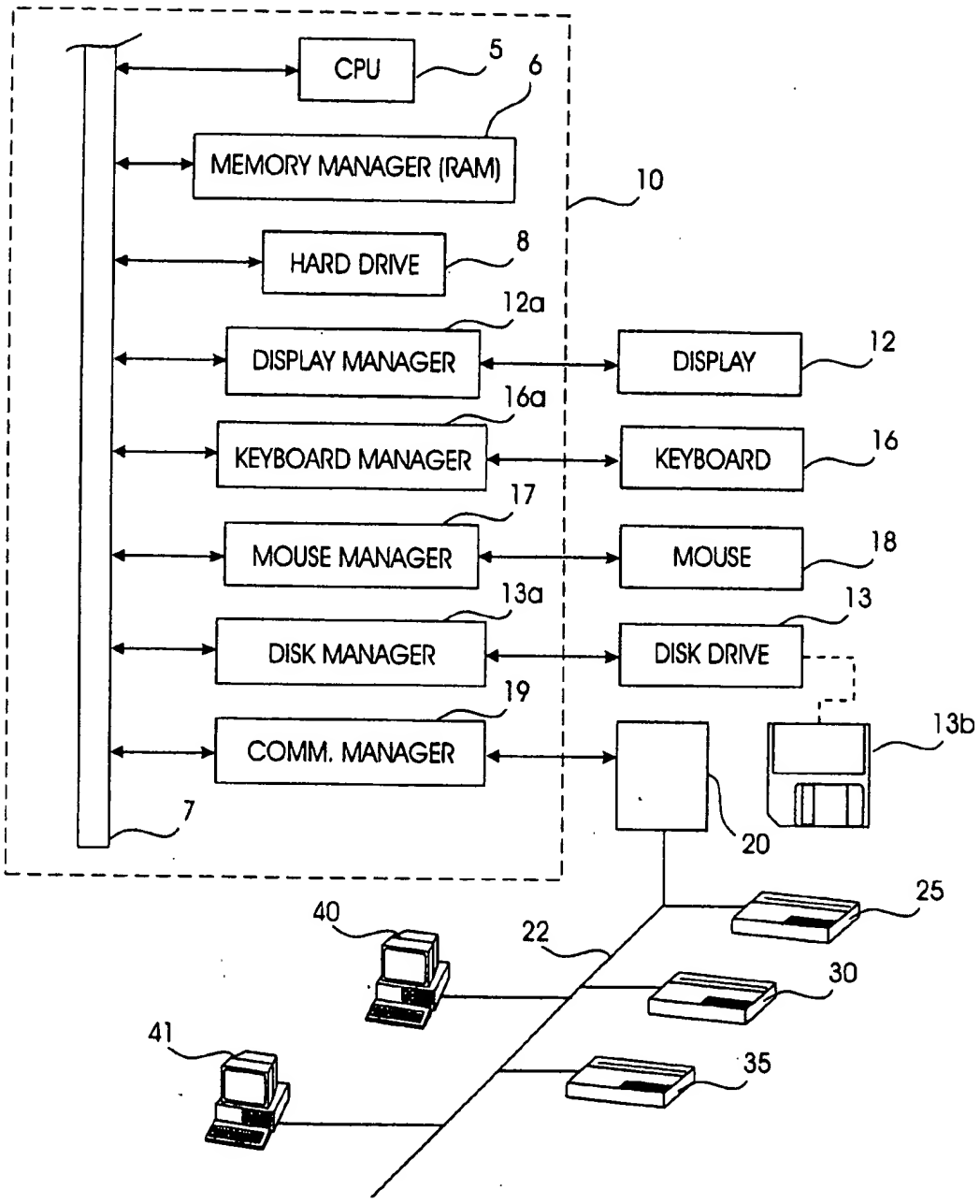

CPU
5
6
MEMORY MANAGER (RAM)
10
HARD DRIVE
8
12a
DISPLAY MANAGER
DISPLAY
12
16a
KEYBOARD MANAGER
KEYBOARD
16
17
MOUSE MANAGER
MOUSE
18
13a
DISK MANAGER
13
DISK DRIVE
19
COMM. MANAGER
13b
7
20
25
40
22
30
41
35

FIG. 2

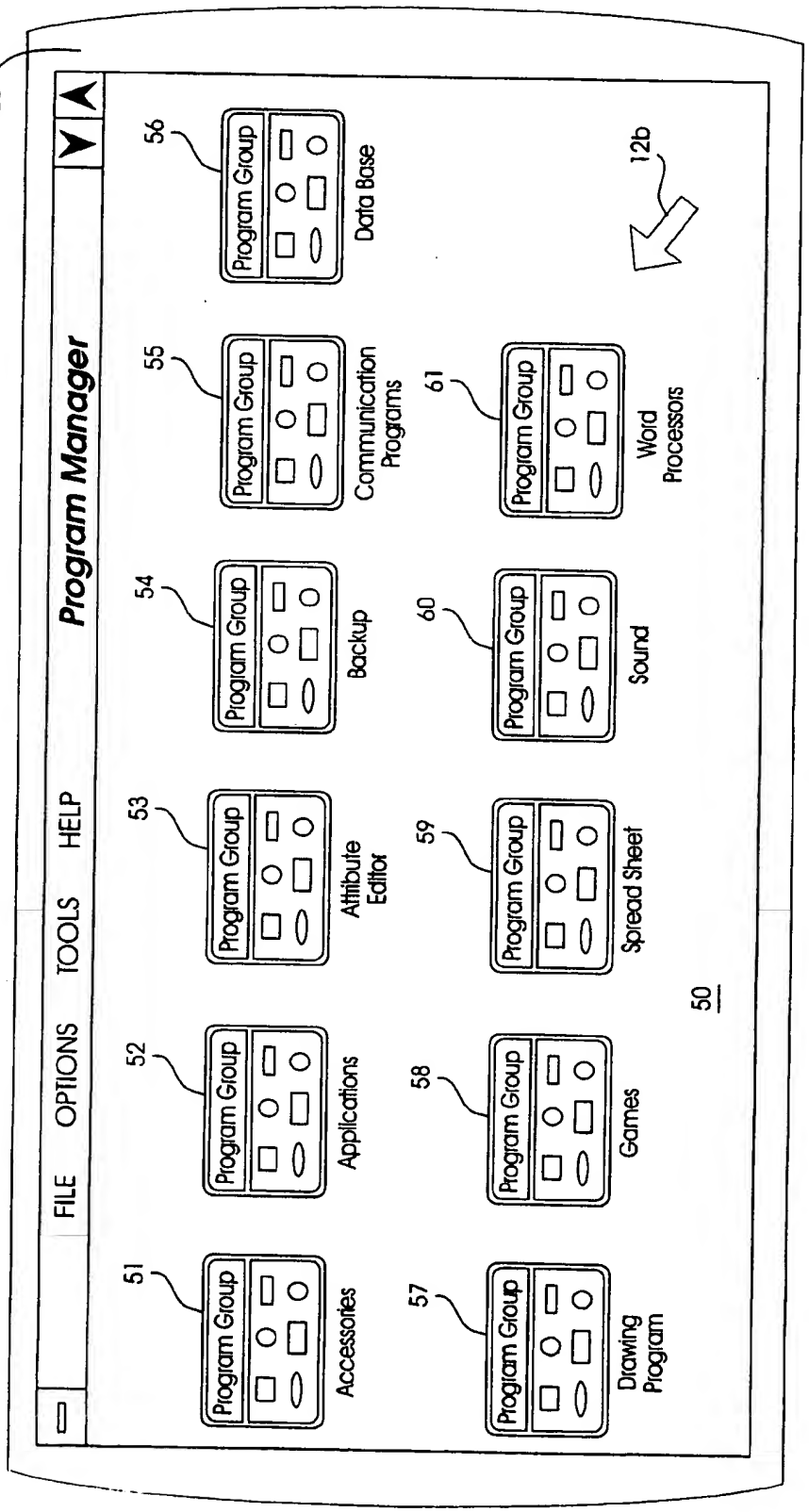

12c
FILE OPTIONS TOOLS HELP
Program Manager
12b
50
51
Program Group
Accessories
52
Program Group
Applications
53
Program Group
Attribute Editor
54
Program Group
Backup
55
Program Group
Communication Programs
56
Program Group
Data Base
57
Program Group
Drawing Program
58
Program Group
Games
59
Program Group
Spread Sheet
60
Program Group
Sound
61
Program Group
Word Processors

FIG. 3

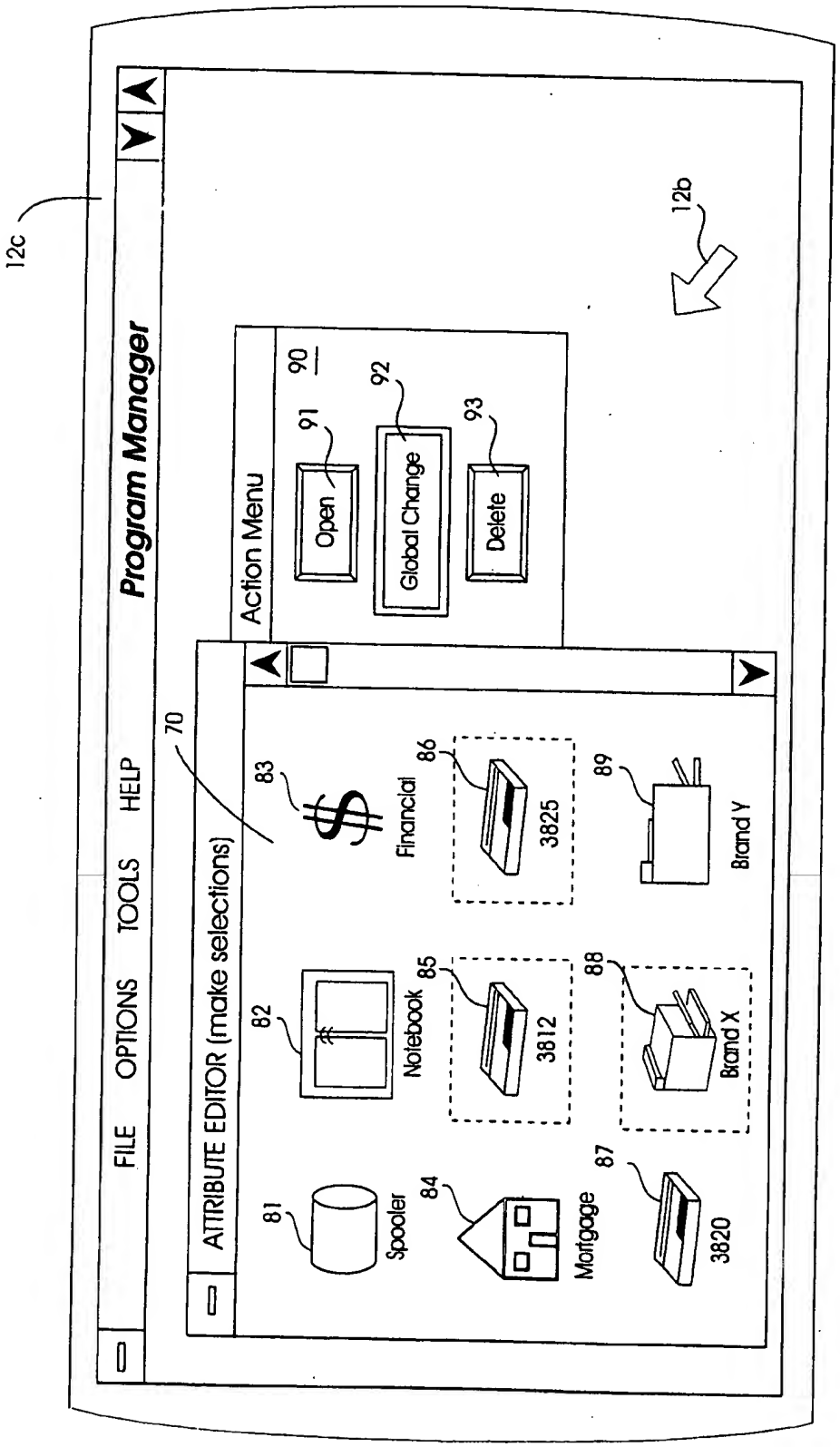

12c
12b
Program Manager
FILE OPTIONS TOOLS HELP
ATTRIBUTE EDITOR (make selections)
70
Action Menu
90
91
Open
92
Global Change
93
Delete
81
Spooler
82
Notebook
83
Financial
84
Mortgage
85
3812
86
3825
87
3820
88
Brand X
89
Brand Y

FIG. 3A

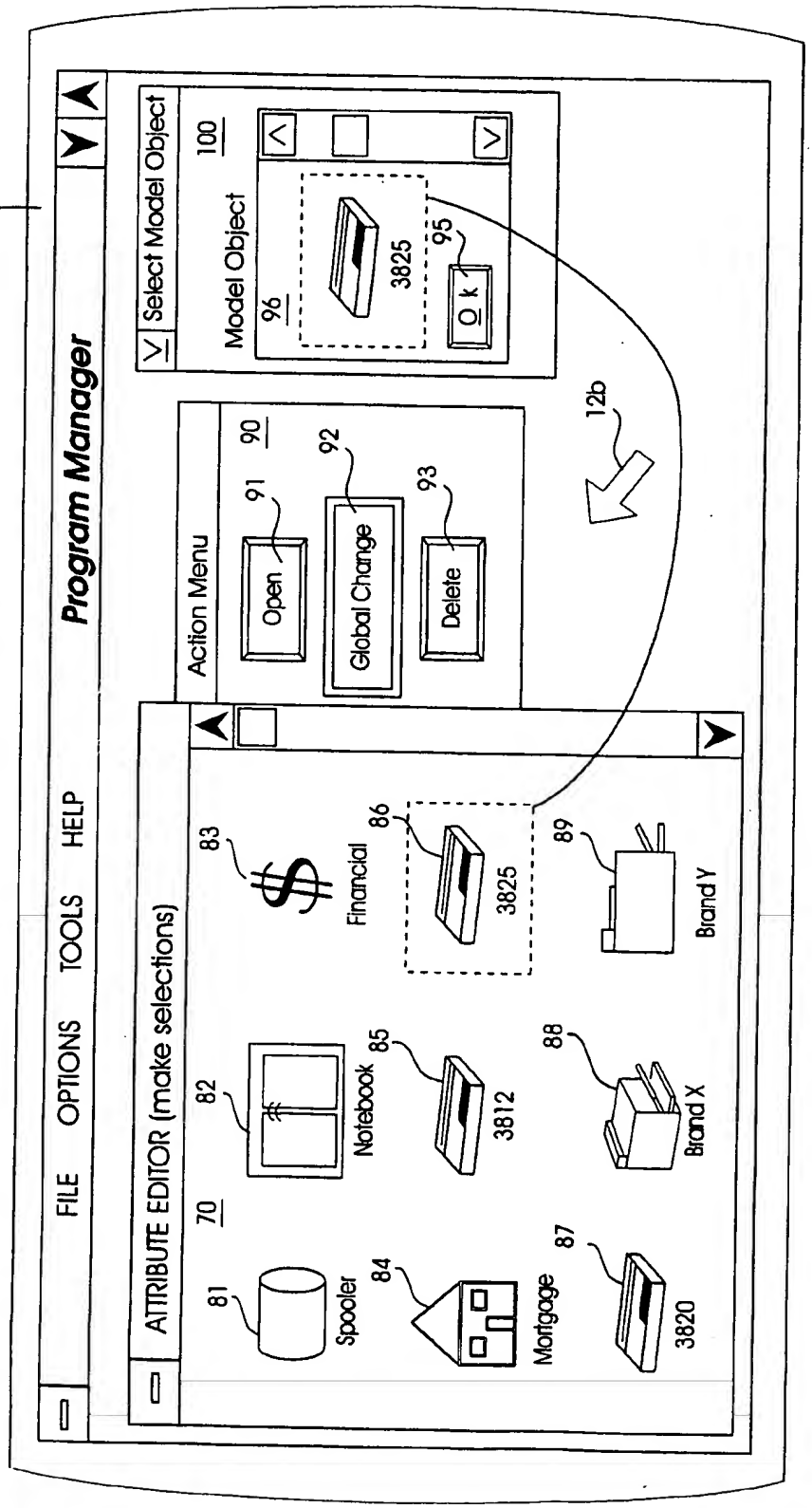

12c
Program Manager
FILE OPTIONS TOOLS HELP
ATTRIBUTE EDITOR (make selections)
70
Select Model Object
100
Model Object
96
3825
95
Ok
Action Menu
90
91
Open
92
Global Change
93
Delete
12b
83
Financial
86
3825
89
Brand Y
82
Notebook
85
3812
88
Brand X
81
Spooler
84
Mortgage
87
3820

FIG. 3B

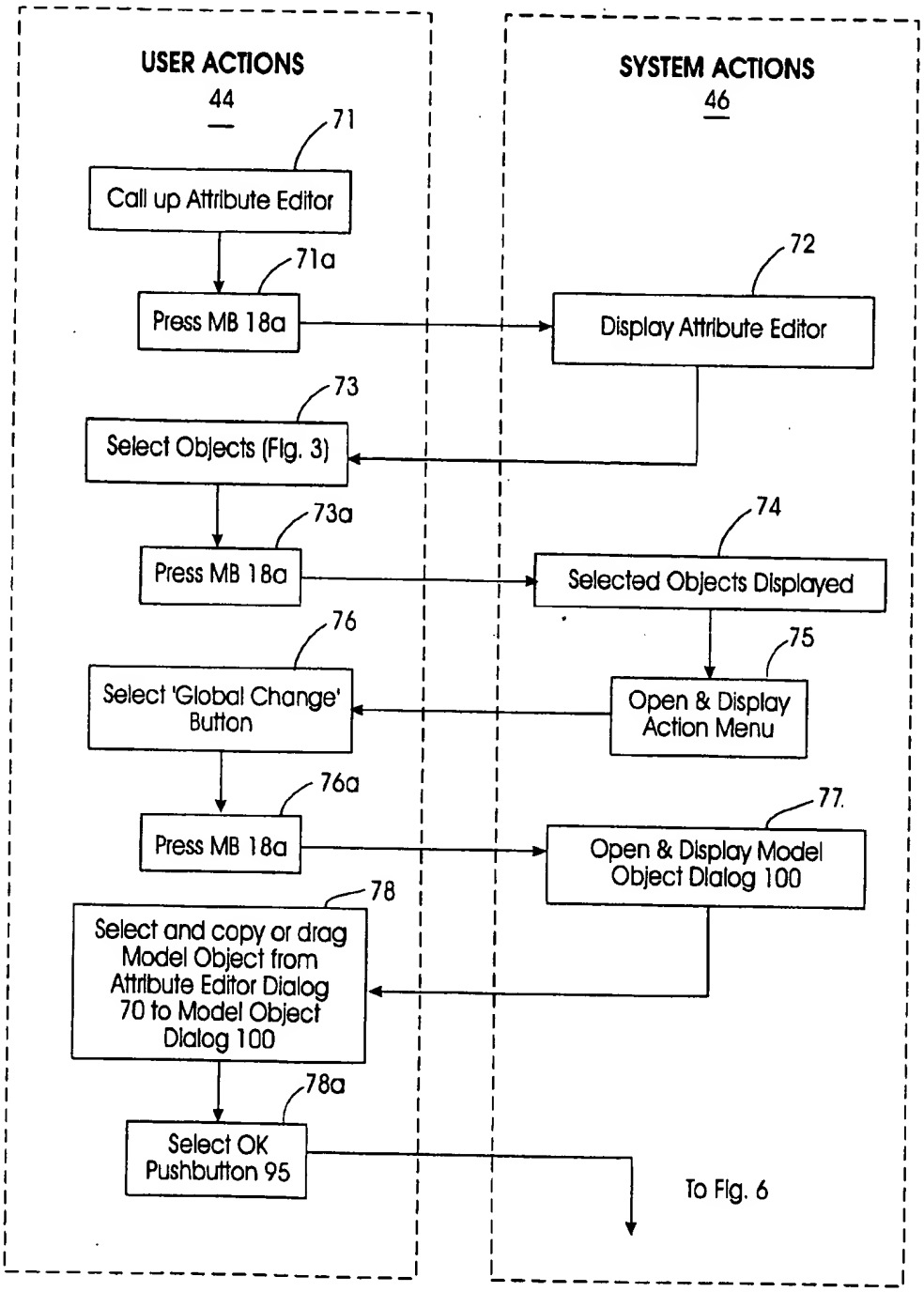

USER ACTIONS
44
SYSTEM ACTIONS
46
71
Call up Attribute Editor
71a
Press MB 18a
72
Display Attribute Editor
73
Select Objects (Fig. 3)
73a
Press MB 18a
74
Selected Objects Displayed
75
Open & Display Action Menu
76
Select 'Global Change' Button
76a
Press MB 18a
77
Open & Display Model Object Dialog 100
78
Select and copy or drag Model Object from Attribute Editor Dialog 70 to Model Object Dialog 100
78a
Select OK Pushbutton 95
To Fig. 6

FIG. 4

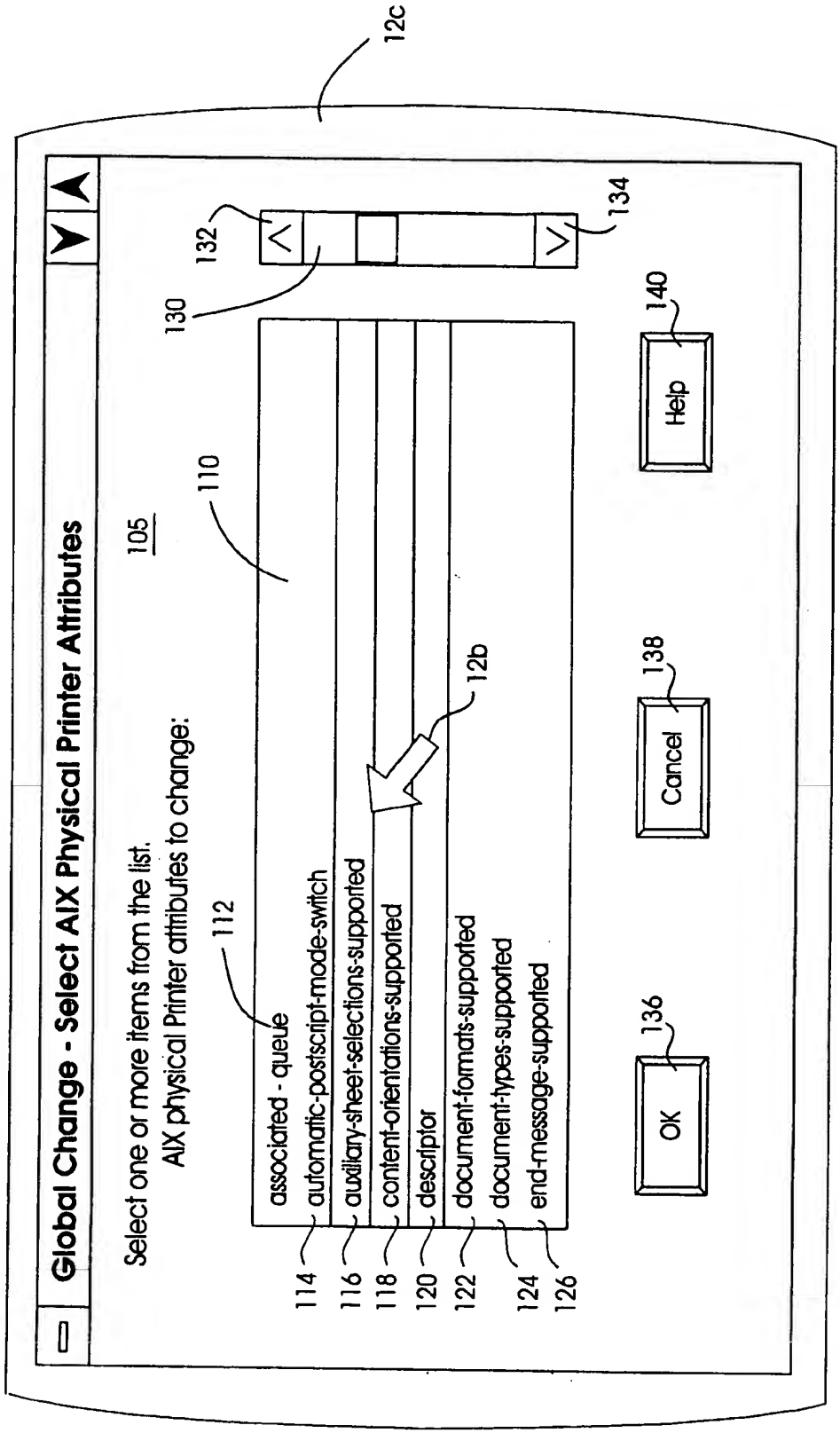

12c
Global Change - Select AIX Physical Printer Attributes
Select one or more items from the list.
AIX physical Printer attributes to change:
105
110
112
130
132
134
140
Help
12b
138
Cancel
136
OK
associated - queue
automatic-postscript-mode-switch
auxiliary-sheet-selections-supported
content-orientations-supported
descriptor
document-formats-supported
document-types-supported
end-message-supported
114
116
118
120
122
124
126

FIG. 5

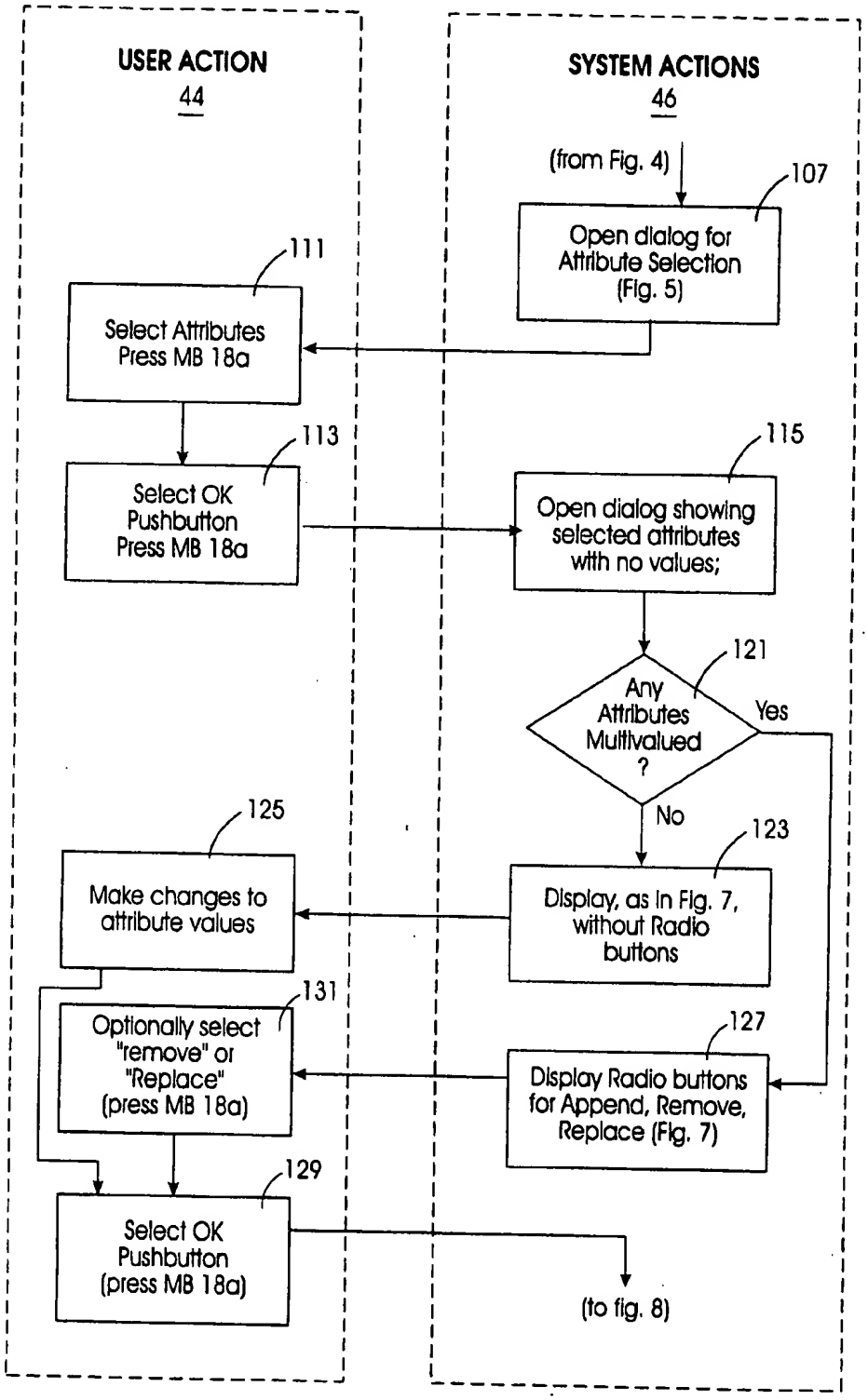

USER ACTION
44
SYSTEM ACTIONS
46
(from Fig. 4)
107
Open dialog for Attribute Selection (Fig. 5)
111
Select Attributes Press MB 18a
113
Select OK Pushbutton Press MB 18a
115
Open dialog showing selected attributes with no values;
121
Any Attributes Multivalued ?
Yes
No
123
Display, as in Fig. 7, without Radio buttons
125
Make changes to attribute values
127
Display Radio buttons for Append, Remove, Replace (Fig. 7)
131
Optionally select "remove" or "Replace" (press MB 18a)
129
Select OK Pushbutton (press MB 18a)
(to fig. 8)

FIG. 6

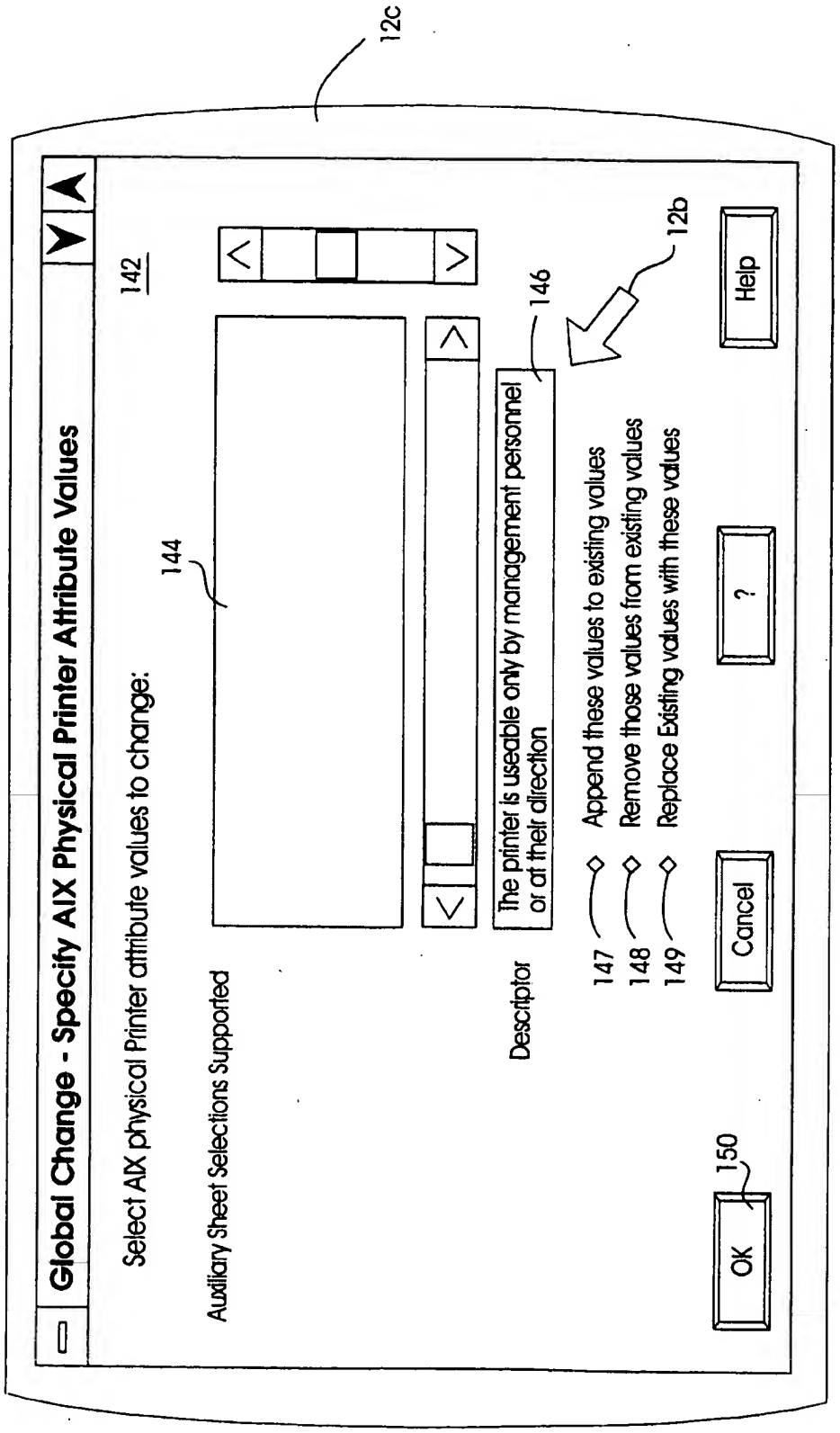

12c
Global Change - Specify AIX Physical Printer Attribute Values
Select AIX physical Printer attribute values to change:
142
144
Auxiliary Sheet Selections Supported
Descriptor
The printer is useable only by management personnel or at their direction
146
12b
Help
147
Append these values to existing values
148
Remove those values from existing values
149
Replace Existing values with these values
?
Cancel
OK
150

FIG. 7

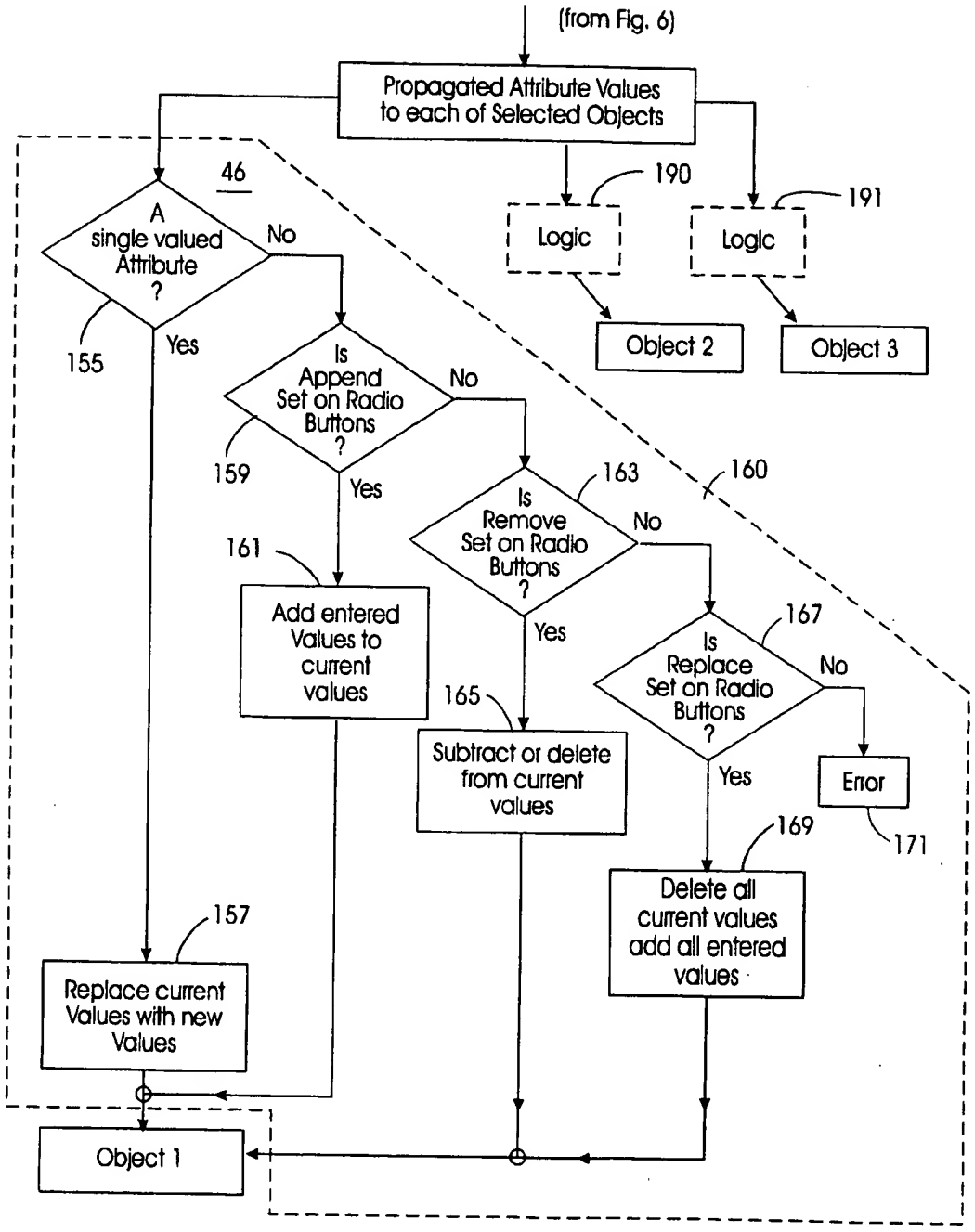

(from Fig. 6)
Propagated Attribute Values to each of Selected Objects
46
A single valued Attribute ?
No
Yes
155
Logic
190
Logic
191
Object 2
Object 3
Is Append Set on Radio Buttons ?
No
Yes
159
Is Remove Set on Radio Buttons ?
163
160
No
Yes
161
Add entered Values to current values
Is Replace Set on Radio Buttons ?
167
No
Yes
165
Subtract or delete from current values
Error
171
169
Delete all current values add all entered values
157
Replace current Values with new Values
Object 1

FIG. 8

METHOD, APPARATUS AND APPLICATION FOR OBJECT SELECTIVE BUT GLOBAL ATTRIBUTE MODIFICATION

BACKGROUND OF THE INVENTION AND STATE OF THE PRIOR ART

1. Field of the Invention

The present invention relates to globally modifying, adding, removing (collectively referred to as "modifying") selected object attributes in designated objects, and more particularly relates to a method, apparatus and application utilizing a graphical interface for enabling such global revision of selected object attributes.

2. Description of Related Art

In computer systems, the user often must spend exorbitant amounts of time on system setup, especially in anticipating his usage of different objects during any one computer session (e.g. day, week, month etc.) Or even for one large job. Often the user must duplicate setups in multiple objects, for example multiple on-line printers used to print from a multitude of applications. This is done so that in any application requiring printing, the attributes associated with each of the printers that may be used, are the same. To continue with the printer example, this kind of setup is particularly critical when the printers are interconnected via a local area network (LAN) or some other type of network and, therefore, each printer is subject to use by a large number of users for printing of multiple documents (or parts of the sane document) requiring the printers to have the same attribute values.

In order to better understand the terms utilized in this patent application, a brief background definition section will be presented so that the reader will have a common understanding of the terms employed and associated with the present invention.

A "user interface" is a group of techniques and mechanisms that a person employs to interact with an object. The user interface is developed to fit the needs or requirements of the users who use the object. Commonly known user interfaces can include telephone push buttons or dials, or pushbuttons such as on a VCR or a television set remote. With a computer, many interfaces not only to allow the user to communicate with the computer but also allow the computer to communicate with the user. These would include (1) command-line user interfaces (i.e., user remembered commands which he/she enters, e.g. "c:\>DIR" in which "DIR" is a typical DOS command entered at the "C" prompt); (2) menu-driven user interfaces which present an organized set of choices for the user, and (3) graphical user interfaces, ("GUI") in which the user points to and interacts with elements of the interface that are visible, for example by a "mouse" controlled arrow or cursor.

An example of a GUI user interface is that which is offered by International Business Machines Corporation (IBM) under the name "Common User Access" ("CUA"). This GUI incorporates elements of object orientation (i.e., the user's focus is on objects and the concept of applications is hidden). Object orientation of the interfaces allow for an interconnection of the working environment in which each element, called an "object," can interact with every other object. The objects users require to perform their tasks and the objects used by the operating environment can work cooperatively in one seamless interface. With objected oriented programming using a GUI, the boundaries that distinguish applications from operating systems are no longer apparent or relevant to the user. While the invention is described in terms of object-oriented interfaces, such an interface is not required for the present invention.

In connection with this patent application, an "object" means any visual component of a user interface that a user can work with as a unit, independent of other items, to perform a task. By way of example, a spreadsheet, one cell in a spreadsheet, a bar chart, one bar in a bar chart, a report, a paragraph in a report, a database, one record in a database, and a printer are all objects. Each object can be represented by one or more graphic images, called "icons," with which a user interacts, much as a user interacts with objects in the real world. (NOTE: In the real world, an object might be an item that a person requires to perform work. As an example, an architect's objects might include a scale, T-square, or a sharp pencil, while an accountant's objects might include a ledger and a calculator.) However, it is not required that an object always be represented by an icon, and not all interaction is accomplished by way of icons. For example, and as will be seen hereinafter, a user can interact with an object by opening a window that displays more information about the object and includes a variety of mechanisms for interacting with the object.

While classification of objects may follow many different definitions, each class of objects has a primary purpose that separates it from the other classes. A class may be looked at as a group of objects that have similar behavior and information structures. In addition, each of the objects enumerated and defined below may contain other objects. There are three primary classes of objects. Each is discussed below.

(1) Container Object: This object holds other objects. Its principal purpose is to provide the user with a way to hold or group related objects for easy access or retrieval. An operating system, e.g. OS/2® (a trademark of IBM Corporation) or Windows® (a trademark of Microsoft Corporation), typically provides a general-purpose container, for example a folder or a program group—that holds any type of object, including other containers. For example, imagine a program group (or folder) labeled "PRIVATE FOLDER—ICONS;". In the program group are three folder icons labeled "REPORT", "PORTFOLIO" and "LETTERS". By selecting with a mouse or other pointing device the icon "PORTFOLIO", another window may open showing three more icons labeled "OIL PAINTINGS", "WATERCOLORS," and "PORTRAITS". In turn, selecting any of those three icons may open additional windows with further icons representing further subdivisions, or cross-references (e.g., "CUSTOMERS").

(2) Data Objects: The principal purpose of a data object is to convey information. This information may be textual or graphical information or even audio or video information. For example, a business report displayed on the computer monitor may contain textual information concerning sales of "gadgets" over the past few years (text object) to all customers and also may contain a bar chart (graphic object) to pictorially depict, on the same monitor screen, the sales information.

(3) Device Objects: The principal purpose of a device object is to provide a communication vehicle between the computer and another physical or logical object. Many times the device object represents a physical object in the real world. For example, a mouse object or icon can represent the user's pointing device, and a modem object can represent the user's modem, or a printer object or icon can represent the user's printer. Other device objects are purely logical, e.g. an out-basket icon representing outgoing electronic mail; a wastebasket object or icon representing a way the user may "trash" or dispose of other objects.

As can be seen from the foregoing, a class of objects may be defined as a description of the common characteristics of several objects, or a template or model which represents how the objects contained in the class are structured. While there are further ways in which to define objects and class of objects, typically each class of objects will include similar attributes, the values of which the user will alter, modify, replace or remove from time to time. (For a more complete discussion of objects, attributes, object oriented interfaces etc., see "Object Oriented Interface Design: IBM Common User Access" (published by Que, ISBN 1-56529-170-0).

As mentioned above, the detail pertaining to objects is provided for background only. Object oriented interfaces are not required for the present invention.

Current graphical interface architectures do not provide a mechanism to facilitate changing the value of an attribute across many (but possibly less than all) objects of the same class. Additionally, there is no effective mechanism for capturing the values of an attribute for one object and propagating the values to other objects of the same class (i.e., making a global change).

There are some word processors and graphical operating systems which afford a similar function, but each has limitations which distinguish it from the solution presented here. For example, Microsoft Word® allows the user to select multiple objects (e.g. paragraphs). Thereafter, with the Format/Paragraph operation, the user is presented with a dialog that allows modified characteristics to be set for each selected paragraph. WordPerfect® for Windows (a trademark of Novell Corporation) provides a method for selecting multiple objects and changing the attributes for all of the objects at the same time but the method, much like that in Microsoft Corporations Words, is restricted to boolean and single valued attributes. OS/2® and Windows® allow multiple file or program objects to be selected and then a single operation (e.g. delete) to be performed on them, but it is limited to a single operation, and cannot be used to change attributes that have multiple values, where the values may be in different formats.

Other attempts have been made to apply global changes to attributes of objects which are of the same name. In U.S. Pat. No. 5,001,654 issued on Mar. 19, 1991 to Wineger et.al, at the choice of the operator a change of the value of properties of a component may be made to apply globally to all components with the same name as the amended component, or be applied solely to the changed component. As shall become apparent from the following description, the present invention differs in that a local value change to an attribute may be propagated to "selected" objects of the same "class", not necessarily all of the objects of the same class. Additionally, if desired, the attributes and their values may be edited for the class and the remaining may then be propagated. Moreover, as also shall become evident, the changes to the values of attributes is permitted to a wide range of attribute types including attributes that have multiple values, where the values may be in different formats.

SUMMARY OF THE INVENTION

In view of the above, it is a principal object of the present invention to provide a graphically oriented method, application and apparatus to facilitate the operator or user in altering attribute values globally to selected objects of the same class.

Another object of the present invention is to permit such alteration globally of a wide range of attribute types including attributes that have multiple values, where the values may be in different formats.

Yet another object of the present invention is to permit the capture of values of selected attributes of a designated object and then propagation of those to selected other objects of the same class.

Still another object of the present invention is to provide an application which may be employed in a number of different computers, may be transported between different computers, and may be loaded into various computer environments.

The invention is carried out in the following environment and utilizing the following briefly set forth method for revision of the value of attributes or the designated properties of selected objects as controlled by a computer system. The computer system has at least a visual operator interface, an operating system for operating applications within the computer system, and memory for storing at least part, preferably all, of an application. The method comprises the steps of: designating objects having properties to be modified and selecting a global change action item. The attributes of the object may then be modified as desired, and the modified attribute values may be propagated globally to the selected objects. Additionally, further attributes from an attribute list associated with the designated objects may be selected, and selected ones of the modified attributes may be propagated globally to the designated objects.

Other objects of the invention and a more complete understanding of the invention may be had by referring to the following description taken in conjunction with the accompanying drawings

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** illustrates a typical computer system which may be employed to practice the novel method and application of the present invention;

FIG. **2** is a block diagram illustrating a sample configuration of the computer system shown in FIG. **1**;

FIGS. **3**, **3***a*, and **3***b* illustrate the "Systems Window" dialog as well as an "Action Menu" dialog as initiated by actions taken by the user and the system to carry out the novel method of the present invention and incorporated as part of the application of the present invention;

FIG. **4** is a block diagram or flow chart illustrating both user actions and system actions to accomplish that which is shown in FIGS. **3**A and **3**B;

FIG. **5** is a typical screen display illustrating yet another step in the novel process of the present invention and utilizing, as an example, printer attributes, selected values of which are to be changed and selection of those desired for such change by the operator;

FIG. **6** is a block diagram or flow chart illustrating both user actions and system actions to accomplish that which is shown in the screen displays of FIGS. **5** and **7**;

FIG. **7** is another screen display illustrating yet another step in the novel process of the present invention;

FIG. **8** is a logic diagram flow chart illustrating how the attributes are changed at the selected objects whether the change is single value, or the change is to be appended, removed, or replaced and regardless of the attribute type.

DESCRIPTION OF THE ILLUSTRATIVE EMBODIMENT(S)

Turning now to the drawings, and especially FIGS. **1** and **2**, FIG. **1** diagrammatically shows a computer system **1** which may be connected to a Local Area Network system

(LAN **20**) as shown in FIG. **2**. As will become more evident from the following discussion, these systems may be employed for globally changing attribute values of selected objects in accordance with the present invention.

As shown in FIG. **1**, the computer system **1** comprises a main chassis **10**, a display means or monitor **12**, a connected keyboard **16** and a pointing device, in the present instance a mouse **18** which is operator controlled to move a pointer cursor **12***b* (shown in FIG. **3**) on the display or monitor screen **12***c*. As shown in FIG. **2**, the chassis **10** includes a central processing unit, or "CPU" **5**, a memory manager and associated random access memory, or "RAM" **6**, a fixed disk or hard drive **8** (which may include its associated disk controller), a display manager **12***a* which is connected externally to the chassis **10** of the display **12**; a keyboard manager **16***a*, which through flexible cable (not shown) is connected to the keyboard **16**; a mouse manager **17** (which in some instances may form part of the display manager **12***a*, and may be in the form of a software driver) for reading the motion of the mouse **18** and its control mouse buttons (MB) **18***a* and **18***b*, shown in FIG. **1**. A disk manager or controller **13***a* which controls the action of the disk drive **13** (and an optional drive such as a magneto-optical or CD ROM drive **14**) shown in FIG. **1**, rounds out most of the major elements of the computer system **1**.

The pointer element or cursor **12***b* can be moved over the display screen **12***c* by movement of the mouse **18**. The mouse buttons (MB) **18***a* and **18***b* give commands to the operating system, usually through a software mouse driver provided by the mouse manufacturer. With the first mouse button (MB) **18***a* the operator can select an element indicated on the display screen **12***c* using the pointer or cursor **12***b*, i.e., signify that an action subsequently to be performed is to be carried out on the data represented by the indicated element on the display screen **12***c*. The system normally gives some visual feedback to the operator to indicate the element selected, such as a change in color, or a blocking of the icon. The second mouse button (MB) **18***b* may be a menu button, if desired. Conventionally, when the operator presses button **18***b*, a selection menu or dialog with system commands will appear can the display screen **12***c*. The operator may select an icon or item from the selection menu or input information into the dialog box as appropriate using the cursor **12***b* and the first mouse button (MB) **18***a*.Some menu items, if selected, may call up another menu or submenu for the operator to continue the selection process.

The use of a mouse and selection menus is well known in the art, for example U.S. Pat. No. 4,464,652 to Lapson et al. describes a selection menu of the pull-down type in combination with a mouse. It should be recognized, of course, that other cursor pointing devices may be employed, for example a joystick, ball and socket, or cursor keys on the keyboard.

The foregoing devices (and software drivers therefore) within the chassis **10** communicate with one another via a bus **7**. To round out the computer system **1**, an operating system (not shown) must be employed. If the computer system is a typical IBM-based system, the operating system may be DOS-based and include a GUI interface such as contained in OS/2®, or WINDOWS®, or other operating system of choice. If the computer system is based upon RISC (reduced instruction set computer) architecture, then the operating system employed may be, in the instance of an IBM-based RISC architectured System/6000®, AIX. Alternatively, if the computer system **1** is a large host computer, such as a an IBM 3090, it may be running an operating system such as MVS or VM. This operating system normally includes a print service facility called PSF, which is a system-wide resource manager, which takes a "job" which has been formatted for a particular printer, and sends certain files, such as fonts, special commands and the like to the printer before sending the file to be printed.

By way of background only, if in the illustrated instance the computer system **1** is a RISC system, such as the IBM RISC/6000® computer system, it may be programmed to send a Page Description Language (PDL) called IPDS (Intelligent Print Data Stream) to a printer. However, other page description languages such as "PostScript" or "PCL" are equally applicable. Such page description languages as "PostScript" or "PCL" are normally, but not always, associated with low-end computer systems, and the present invention is equally applicable to those languages. The PDL sets the format of the page to be printed (i.e., start at 1" down and 1" to the right of the upper left corner of the page) and sends it to a rasterizer which takes the description, interprets it, error checks it and builds a pagemap which is logically composed of a two-dimensional array of bits, sometimes referred to as a bitmap. The pagemap is then rearranged and the words are transformed to a page in row major order. The words are then sent to the printer, and as is conventional, the printer printhead receives the words sent to it in a manner that allows it to print across the paper.

In the illustrated instance, the computer system **1** includes an I/O (Input/Output) manager or communications manager **19** (shown in FIG. **2**) which serves to link the computer system for communications with the outside world such as to a systems printer, a modem or a LAN controller (such as a Token ring or ETHERNET or even through a modem employing SDLC) such as shown at **20** in FIG. **2**. The LAN controller may be incorporated inside the computer system **1** or located externally as shown diagrammatically in FIG. **2**, as desired. The LAN controller **20** may connect to other computer systems **10** and **41** as well as to other printers such as printers **25**, **30** and **35** by communications cable **22** and the like. However the method and application of the present invention works equally well with multiple objects serviced by a single computer system or with multiple objects such as printers or servers which service the computer system or systems. Moreover, the principals of operation make the operation of global modification to attributes of selected objects, applicable to multiple objects on a single computer system.

Assume that the operator or system user decides to change the values of selected attributes of some objects within his control. By way of a first example, assume that the system operator has the capability of interconnecting, through the LAN **20**, other computer systems and printers, such as described with reference to FIG. **2**, and it is desired to change certain attributes of multiple printers attached to the LAN **20**.

As is conventional, when the user desires to open one, or more windows, the mouse **18** is moved until the pointer **12***b* is in contact with the desired icon. Depending upon the operating system, simply depressing the left mouse button **18***b* may act to select and open the program group. In other operating systems, rapidly depressing the left mouse button twice selects and opens the program group. In either instance, opening the icon displays icons therein representative of certain programs which have been collected within the particular icon. Throughout this discussion, however, assume that depressing the mouse button **18***a* effects the desired action, regardless of whether it takes one or two depressions. Moreover, iconic representations of the same program or data may appear in more than one icon, or may

be associated with more than one program within more than one group. For example, a particular printer may appear associated with programs in a data base icon **56**, word processor icon and spread sheet icon.

As shown in FIG. **4**, and to facilitate an understanding of the "User Actions" and "System Actions" carried out by the present invention, the flow charts of FIGS. **4** and **6** are thus partitioned as by light dashed lines to form boxes **44** and **46** respectively. As implied by their titles, the user sits in front of the display screen **12***c*, taking the actions set forth in the logic boxes in box **44**, while the system takes the actions, as a result of the user actions, set forth by there logic boxes in box **46**. The user may call up the Systems Window by placing the pointer cursor **12***b* on the System s Window icon (not shown) and pressing the first mouse button (MB) **18***a*, which opens the Systems Window **70**.

Referring now to FIG. 3A, the display screen **12***c* of the monitor **12** displays the labeled "Systems Window" dialog **70**. As illustrated, and by way of example only, the Systems Window dialog **70** shows a plurality of icons **81–89** respectively, which represent under the CUA classification scheme, either device objects, such as spooler **81**, printers **85–89**; data objects, such as notebook **82**, or container objects, such as financial **83** or mortgage **84** packages. It is recognized that each of the classifications may contain other objects which can be classified in other of the CUA object classifications or even classes that the user defines for his or her own purposes.

In accordance with the invention, the user selects the objects, the attributes of which it is desired to modify, add, append, remove or replace (hereinafter collectively referred to as "modify"), and selects the global change button **92**. After the attribute values are modified as desired, the attribute value modifications are propagated to the selected or designated objects.

To this end, assume that the user selects three objects for modification of their attribute values. Of the five printers represented as icons **85–89**, assume that the user selects printers **85**, **86** and **88**. (An IBM Model 3812, IBM Model 3825, and a Brand X printer, respectively), by placing the cursor pointer **12***b* on each of the desired printers **85**, **86**, or **88** and depressing mouse button **18***a*. With respect to FIG. **4**, the forgoing encompasses logic steps **73** and **73***a*. The selection is indicated or displayed, in the present instance, by the light dashed line circumscribing the icons **85**, **86** and **88** (logic step **74**, FIG. **4**). It should be recognized that any convenient means may be employed for indicating user selection, such as color change. As illustrated in FIG. **4** in logic step **75**, this results in opening of an action menu dialog **90** with three on-screen buttons, **91–93** for opening an icon with button **91**, for deleting an icon as with button **93** and of greater concern, a global change button **92**. The global change button **92** is then selected by placing the cursor pointer **12***b* on button **92** and depressing mouse button **18***a*.

It is important to recognize that the steps of the present invention discussed above, may be arranged in any convenient order. For example, global change could be selected prior to the selection of designated objects. Moreover, modification to the attribute values of a model object could be made before selection of global change or the designation of objects to which the modified attribute values apply. Clearly, however, the invention does require at some point, the selection of a model object, the designation of objects to which the modification must be made, a global command for application of the modified attributes to the designated objects, the modification of the attributes of the model or template and the propagation of the modifications to the designated objects.

Turning now to FIG. **5**, and assuming that what is desired is to modify certain physical printer attributes of the printers **85**, **86** and **88** (FIG. 3A), and that in the illustrated instance the printers are operating under control of computer systems utilizing an AIX operating system. The dialog **105** contains a window **110** with a typical vertical scroll bar **130**, containing conventional up and down pointers **132** and **134** respectively, for controlling, upon selection by the cursor pointer **12***b* under mouse **18** control, a scrollable list **112** of printer attributes. Several of these attributes are listed in the window **110**. By way of example only, the number corresponding to the identification number (#) in FIG. **5**, the meaning of these attributes, whether they are single valued (SV), multivalued (MV), boolean (BOO) or complex (COM) is set forth in the table below:

#	Attribute	Description	Value
112	associated-queue	Job Queues, by name	SV
114	automatic-postscript-mode-switch	Description of printer operation. Yes/No, query	BOO, or SV
116	auxiliary-sheet-selections supported	job sheets in front of job; separator sheets for job, sheet color etc.	MV
118	content-orientations-supported	Landscape, portrait, reverse landscape etc.	MV
120	descriptor	descriptive text string by user to give some message to other system users	SV
122	document-formats-supported	ASCII, PostScript, PPDS, PCL etc.	MV
124	document-types-supported	overlays, page segments	MV
126	end-message-supported	job complete	BOO, SV

Assuming, for example, that the user wishes to modify the attribute **116** (auxiliary-sheet-selections-supported) and attribute **120** (descriptor). These two items are selected by the mouse **18** and the cursor pointer **12***b*, and are indicated by logic steps **107** and **111** respectively in FIG. **6**, and the lines surrounding the selected items in the list **110** of dialog **105**.

As shown in FIG. **5**, the user may cancel the operation, if desired by selecting the cancel button **138**, ask for help by selecting the help button **140**, or if the user decides to continue, by selecting the OK button **136**. (See logic step **113** in FIG. **6**). Selecting the OK button **136** results in the system opening dialog **142**, FIG. **7**, showing selected physical printer attribute values to change, with no values depicted in the window **144** for attribute **116** "auxiliary sheet selection supported," nor in the window **146** for the attribute **120** "descriptor." The attributes may be keyboard entry by the user, and whatever they may be, the individual ones may be tagged by appropriate selection of the radio buttons **147**, **148** and **149**. These buttons serve respectively to append, remove or replace the values. The logic that serves to prepare the modified attribute values for propagation is depicted as logic steps **121**, **123**, **125**, **127**, **129** and **131**. After the values have been tagged, the OK button **150** may be selected, again as by cursor pointer **12***b* as operated by the mouse **18** and mouse button **18***a*.

Upon selection of the OK button **150** using the mouse as in logic step **129**, the system propagates the attribute values

to each of the selected objects, in the illustrated instance the selected printers. In FIG. 8, the printers have been relabelled object 1, 2 and 3, each of the objects including associated software or hardware logic control to permit modifying selected existing values in the attribute list associated with each of the objects. One particular logic scheme **160** is shown in dashed lines around the logic boxes connected to the object 1. The logic schemes associated with objects 2 and 3 and labeled **190, 191** respectively, are identical to logic scheme **160**, hereinafter described.

The logic scheme **160** may be arranged in any convenient manner. In software, an if-then-else scheme for handling "append," "remove" or "replace" may conveniently be utilized. Other schemes, such as a "decision tree" scheme depicted in logic scheme **160** may be utilized. Either scheme will work equally well and the scheme illustrated is only for ease of understanding. As shown in FIG. **8**, the propagated attribute values go first to decision logic **155** to first ascertain whether the attribute is single-valued. If yes, the current value of the attribute is replaced as at logic block **157**. If the attribute value is not single, the next decision logic block **159** determines which one of the radio buttons is set on (FIG. **7**). In the present instance since the append button **147** is set on, the value is added, as at logic block **161**, to the current value of the attribute of object 1. If the append button is not selected, then the decision logic block **163** is queried to determine if another radio button, in the illustrated instance the "remove" radio button **148** has been selected. If it has, then the value is subtracted or deleted from the selected objects. If the remove button is not selected, then decision logic **167** asks if the "replace" radio button **149** (FIG. **7**) has been selected. If so, all current values associated with the existing attribute will be deleted and replaced by the new or changed values. This latter step is illustrated in logic box **169**.

An alternative embodiment of the invention utilizes a model object dialog box. Referring now to FIG. **9**, suppose the user has selected the objects of interest and has selected the global change button **92** as described with reference to FIG. **3A**. The user now is prompted for and selects a model object from the collection of objects selected. As shown in FIG. **9**, suppose that of the three printers selected, the user desires the IBM 3825 printer object **86** to be the model object. This selection is reflected in FIG. **9** in window **100**. The model object **96** is IBM printer 3825. The system then displays the attribute options and the presently selected attributes for the selected model object **96**. The user then modifies the attributes as desired and, upon completion, selects a pushbutton labeled "OK" or "propagate changes." The attribute modifications then are propagated to all of the selected objects.

Of course, other means may be employed to identify the model object for attribute value modification. For example, after selection of the model printer object **86**, the icon may be copied to the model object dialog window **96** by utilization of the conventional pull down FILE menu found in most operating systems. Alternatively, the object dialog window **96** may contain a scrollable listing, either by name or icon, of objects contained in the attribute dialog **70** with one object being capable of being selected. Regardless of scheme, after the model or model object is selected, selection by the user of the OK button **95** using the mouse **18**, as at logic step **78***a* (FIG. **4**) causes the system to open dialog **105**, (FIGS. **5** and **6**).

Although the invention has been described with a certain degree of particularity, it should be recognized that elements thereof may be altered by person(s) skilled in the art with out departing from the spirit and scope of the invention. The invention is limited only by the following claims and their equivalents.

What is claimed is:

1. An application for revision of the value of attributes of selected objects as controlled by a computer system having at least a visual operator interface, an operating system for controlling the operation of program applications within the computer system, and memory for storing a program application, the application comprising:

means for designating objects, having attributes to be altered, modified, added or deleted (collectively, "revised");

means for selecting a global change operation for all of the designated objects having attributes to be revised;

means for revising the attributes as desired; and

means for propagating the revised attributes to the designated objects.

2. An application for revision of the value of attributes of selected objects as controlled by a computer system having at least a visual operator interface, an operating system for controlling the operation of applications within the computer system, and memory means for storing an application, the application comprising:

means for selecting a model object;

means for designating objects having attributes to be revised;

means for selecting a global change operation for all of the designated objects having properties to be revised;

means for revising attributes of the model object; and

means for propagating the revised attributes to the designated objects.

3. An application for revision of the value of attributes of selected objects as controlled by a computer system having at least a visual operator interface, an operating system for controlling the operation of program applications within the computer system, and memory for storing a program application, the application comprising:

means for designating objects, having attributes to be altered, modified, added or deleted (collectively, "revised");

means for selecting a global change operation for all of the designated objects having attributes to be revised;

means for selecting a model object from the designated objects;

means for revising attributes of the model object; and

means for propagating the revised attributes to the designated objects.

4. A method for revision of the value of attributes of selected objects as controlled by a computer system having at least a visual operator interface, an operating system for controlling the operation of applications within the computer system, and memory for storing at least part of an application, the method comprising the steps of:

designating objects having attributes to be altered, modified, added, or deleted (collectively, "revised");

selecting a global change operation for all of the designated objects having attributes to be revised;

revising the attributes as desired; and

propagating the revised attributes to the designated objects.

5. A method for revision of the value of attributes of designated properties of selected objects as controlled by a computer system having at least a visual operator interface, an operating system for controlling the operation of applications within the computer system, and memory means for storing an application, the method comprising the steps of:

selecting a model object;

designating objects having attributes to be revised;

selecting a global change operation for all of the designated objects having properties to be revised;

revising attributes of the model object; and

propagating the revised attributes to the designated objects.

6. A method for revision of the value of attributes of selected objects as controlled by a computer system having at least a visual operator interface, an operating system for controlling the operation of applications within the computer system, and memory for storing at least part of an application, the method comprising the steps of:

designating objects having attributes to be altered, modified, added, or deleted (collectively, "revised");

selecting a global change operation for all of the designated objects having attributes to be revised;

selecting a model object from the designated objects;

revising attributes of the model object; and

propagating the revised attributes to the designated objects.

* * * * *

US006038395A

United States Patent [19]

Chow et al.

[11] Patent Number: 6,038,395

[45] Date of Patent: Mar. 14, 2000

[54] **SYSTEM AND METHOD FOR IMPLEMENTING PROXY OBJECTS IN A VISUAL APPLICATION BUILDER FRAMEWORK**

[75] Inventors: **ChungHen Chow; James William Fountain, Jr.; Rene Lim Llames**, all of Austin, Tex.

[73] Assignee: **International Business Machines Corporation**, Armonk, N.Y.

[21] Appl. No.: **08/357,838**

[22] Filed: **Dec. 16, 1994**

[51] **Int. Cl.**[7] **G06F 9/45**

[52] **U.S. Cl.** **395/702**; 395/702; 395/703; 395/710; 395/701

[58] **Field of Search** 395/701, 685; 709/305; 345/967

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,860,204	8/1989	Gendron et al.	395/702
4,943,932	7/1990	Lark et al.	706/60
5,067,072	11/1991	Talati et al.	395/708
5,262,761	11/1993	Scandura et al.	345/133
5,315,703	5/1994	Matheny et al.	345/507
5,315,709	5/1994	Alston, Jr. et al.	707/6
5,339,433	8/1994	Frid-Nielsen	395/705
5,485,600	1/1996	Joseph et al.	395/500.34
5,487,141	1/1996	Cain et al.	345/435
5,497,463	3/1996	Stein et al.	395/500.41
5,517,663	5/1996	Kahn	345/473
5,613,148	3/1997	Bezviner et al.	395/701
5,642,511	6/1997	Chow et al.	395/701
5,737,607	4/1998	Hamilton et al.	395/701
5,842,017	11/1998	Hookway et al.	395/707
5,848,273	12/1998	Fontana et al.	395/701
5,857,191	1/1999	Blackwill, Jr. et al.	705/35
5,862,052	1/1999	Nixon et al.	395/651
5,862,325	1/1999	Reed et al.	395/200.42
5,867,647	2/1999	Haigh et al.	395/726
5,872,973	2/1999	Mitchell	395/702

OTHER PUBLICATIONS

"Visual Script Action Objects", IBM Technical Disclosure Bulletin, vol. 37, No. 04A, Apr. 1994, p. 683.

"Expose Connection List as Property", IBM Technical Disclosure Bulletin, vol. 37, No. 08, Aug. 1994, p. 433.

"Attribute Action Objects", IBM Technical Disclosure Bulletin, vol. 37, No. 02B, Feb. 1994, p. 665.

"PRoxy Definition Tool for Visual Builder", IBM Technical Disclosure Bulletin, vol. 37, No. 04B, Apr. 1994, p. 347.

"User Interface for Scripts in an OO Visual Builder", IBM Technical Disclosure Bulletin, Vo. 37, No. 04B, Apr. 1994, p. 449.

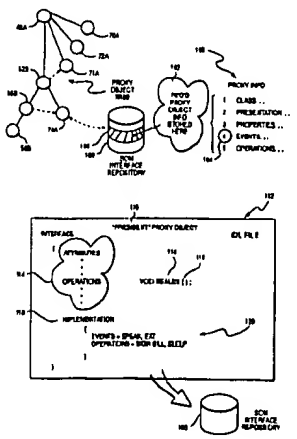
(List continued on next page.)

Primary Examiner—James P. Trammell
Assistant Examiner—Cuong H. Nguyen
Attorney, Agent, or Firm—Robert M. Carwell

[57] **ABSTRACT**

An object model-based visual builder is provided which includes proxy objects at build time, each corresponding to a target object at runtime. Each such proxy object has associated therewith several items of information, including the class of the target object corresponding to the proxy object, presentation information, properties and how they are mapped to IDL attributes and operations, events available on the target object, and operations supported by the target object. A portion of such information is stored in an Interface Repository, such as the System Object Model (SOM) Interface Repository, in easily changeable form, with the necessary knowledge for retrieving such information from the Interface Repository being contained in the proxy object itself. The information stored in the Interface Repository may be changed without altering or recompiling the source code which implements the proxy object itself. Interface declarations for proxy objects are provided in corresponding IDL files stored in the Interface Repository. Implementation statements in the files permit modifiers which encode the easily editable portions of the proxy object information.

32 Claims, 4 Drawing Sheets



OTHER PUBLICATIONS

"Lock Icon to Secure Visual Builder Controls", IBM Technical Disclosure Bulletin, vol. 37, No. 04B, Apr. 1994, p. 21.

Dave et al, "Proxies, Application Interfaces, and Distributed Systems", Object Orientation in Operating Systems, pp. 212(10), Sep. 1992.

Lau-Kee et al, "VPL: An Active, Declarative Visual Programming System", Visual Languages, 1991 IEEE Workshop, pp. 40(8), Oct. 1991.

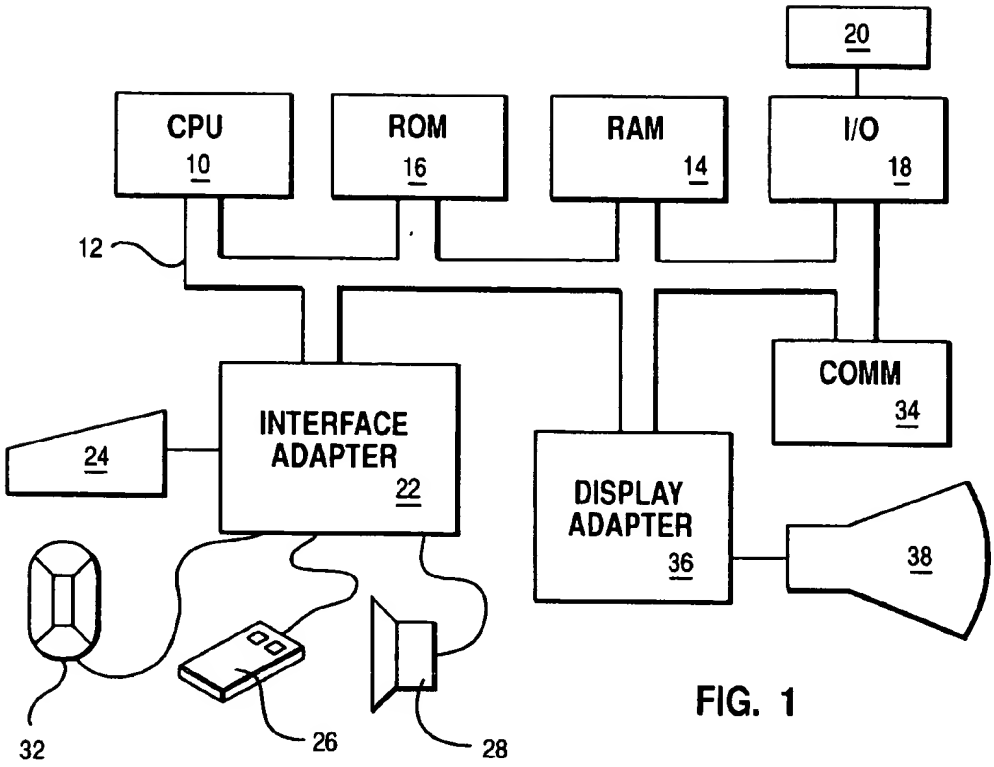

20
CPU
10
ROM
16
RAM
14
I/O
18
12
COMM
34
INTERFACE
ADAPTER
22
24
DISPLAY
ADAPTER
36
38
32
26
28

FIG. 1

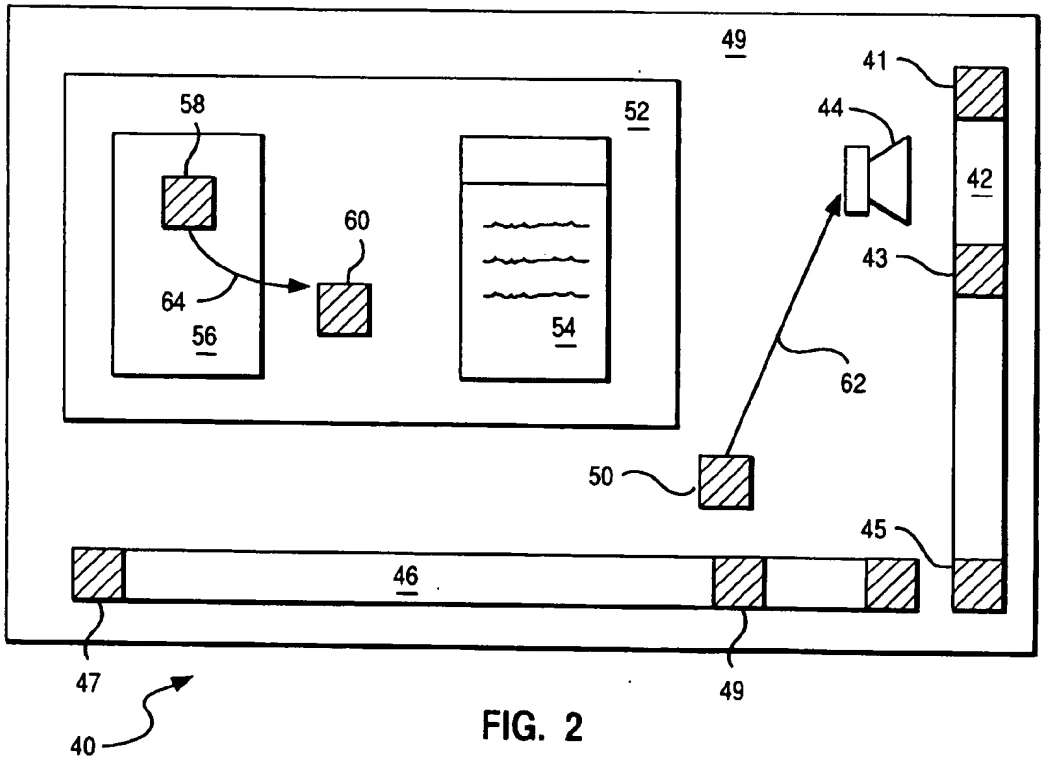

49
41
58
52
44
42
60
43
64
56
54
62
50
45
46
47
49
40

FIG. 2

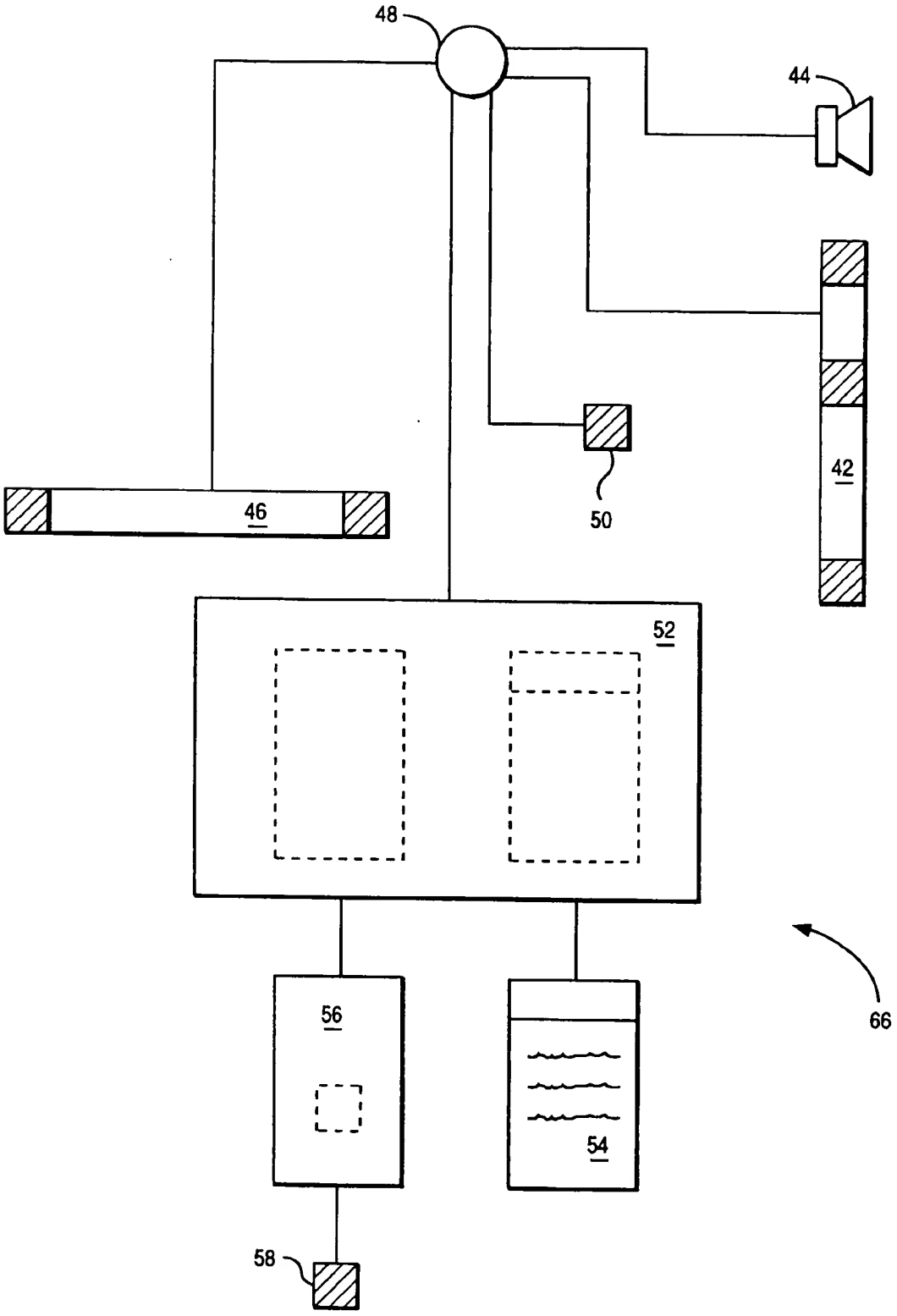


FIG. 3

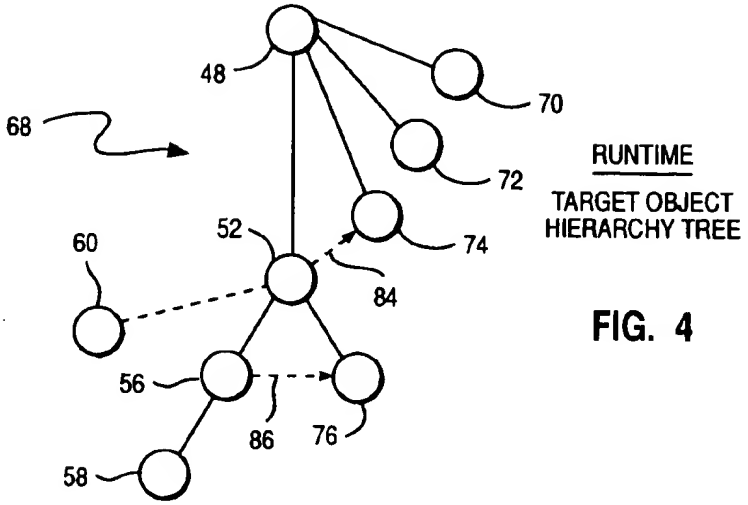


FIG. 4

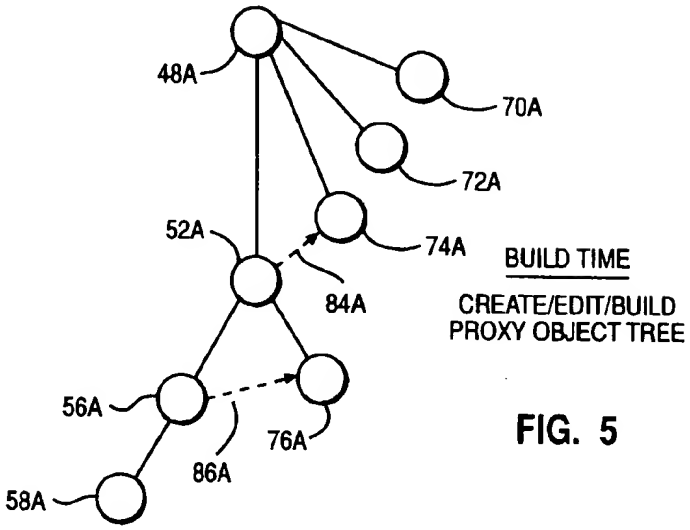


FIG. 5

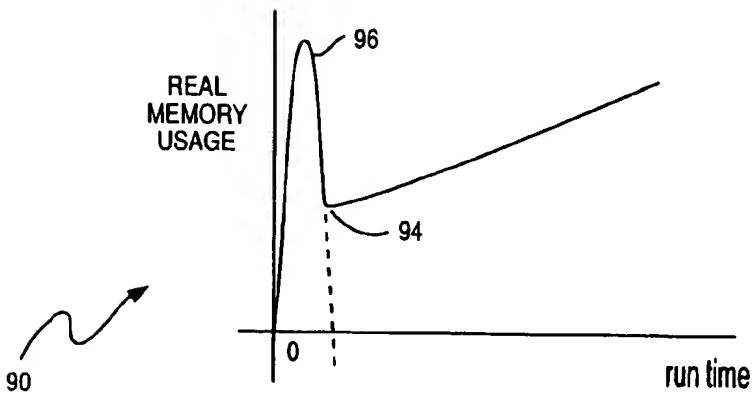


FIG. 6

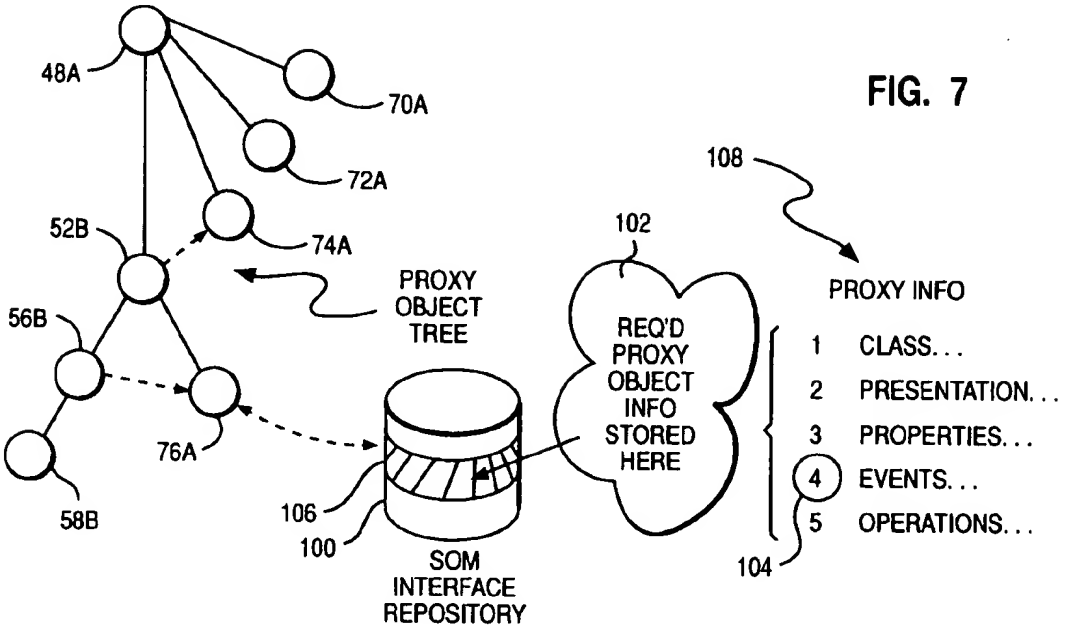

48A
70A
72A
74A
52B
56B
76A
58B
PROXY OBJECT TREE
106
100
SOM INTERFACE REPOSITORY
102
REQ'D PROXY OBJECT INFO STORED HERE
108
PROXY INFO
1 CLASS...
2 PRESENTATION...
3 PROPERTIES...
4 EVENTS...
5 OPERATIONS...
104

FIG. 7

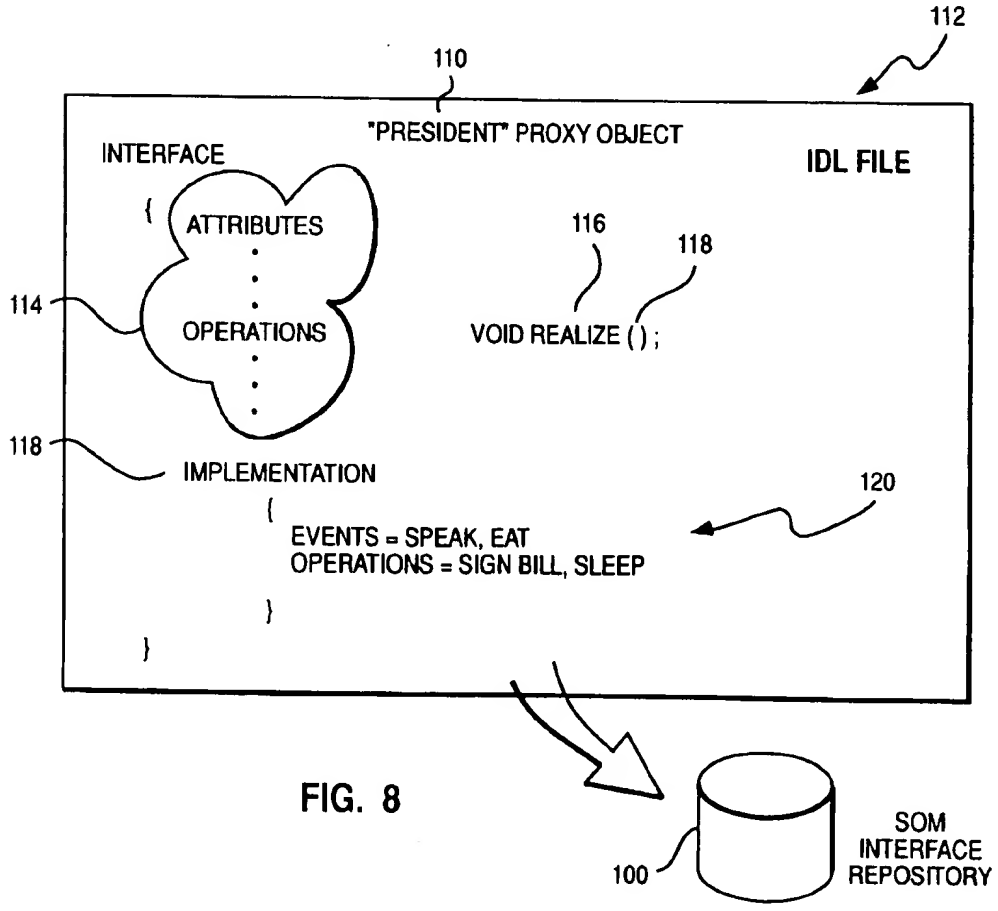

112
110
"PRESIDENT" PROXY OBJECT
INTERFACE
IDL FILE
{
ATTRIBUTES
OPERATIONS
114
116
118
VOID REALIZE ();
118
IMPLEMENTATION
{
EVENTS = SPEAK, EAT
OPERATIONS = SIGN BILL, SLEEP
}
}
120
100
SOM INTERFACE REPOSITORY

FIG. 8

SYSTEM AND METHOD FOR IMPLEMENTING PROXY OBJECTS IN A VISUAL APPLICATION BUILDER FRAMEWORK

CROSS-REFERENCE TO RELATED APPLICATIONS CASES

The present application is related to U.S. patent application Ser. No. 08/357,834, entitled "SYSTEM AND METHOD FOR PROVIDING A VISUAL APPLICATION BUILDER FRAMEWORK", filed on Dec. 16, 1994, which is herein incorporated by reference.

TECHNICAL FIELD

This invention generally relates to computerized object-oriented technology and, more particularly, to improve interactive user interface development systems and methods.

BACKGROUND OF THE INVENTION

Prior to the development of object-oriented programming, a form of programming was developed known as structured programming, which is still used extensively. In this technique, various functions are first defined, and the program then consists of calling such defined functions at appropriate times to effect the overall objective of the application program. Structured programming provided the opportunity of a modularized approach—a significant improvement over "spaghetti code" which was difficult to debug and maintain. Nevertheless, several drawbacks to structured programming remain, such as the "semantic gap" between the concepts and entities in the world being modeled by the program and the concepts in the programming language, the textual nature of the program code, and limitations on reusability of code modules.

Eventually, a new programming paradigm was developed, referred to as object-oriented programming. In this technique, instead of defining functions, "objects" are defined (by defining their "class"). Only the external interfaces of the objects and what can be done with the objects are specified for the end-user, as opposed to the internals of they are constructed.

As a simplistic example of this technique of describing the external "appearance" of an object without the necessity of describing what it "looks like" or "does" inside, and in order to illustrate the significant benefits of such object-oriented programming, one might imagine, by analogy, a common household toaster, which could be modeled by the object-oriented approach. The external "interface" would consist of the knob, which essentially represents a parameter for adjusting the desired lightness or darkness of the toast, the slot into which the bread is inserted, and a lever for effecting the insertion and activating the toaster. The important concept is that the end-user of such an appliance does not need to know the "internals" of the toaster in order to use it, e.g. whether heat is provided by electricity, or by chemical or other means. All the user would have to do is interface properly with these externals.

The beauty of this evolution in programming is that, continuing with the analogy, the designer of the object, which might be a module of programming code modeling a toaster, may focus on improving the internals of the toaster, making it more efficient, for example, without affecting the user's ability to use the object, inasmuch as the internals are transparent to the user. Thus, more abstractly, object-oriented technology may be seen essentially as providing the

advantage of separation of the interface of an object from it implementation. In a software context, the internals of the object may thereby be rewritten and improved without the necessity of rewriting the entire application program, as long as the external "knobs" etc, and their expected behavior have not changed.

The foregoing illustrates one of two important characteristics of object-oriented technology, namely, that of "encapsulation". The other characteristic is "inheritance", whereby, broadly speaking, an object may "inherit" (or acquire) all or some of its interface or implementation from another similar object, thereby avoiding the need to duplicate the descriptions of such common characteristics. For example, a "WheeledCarrier" class-of-objects may be defined, having the generic characteristics and definitions of having wheels and being able to carry passengers. A "Car" class of objects can then be defined, which inherits part of its description from "WheeledCarrier". Similarly, an "Airplane" class of objects can also be defined, which inherits from "WheeledCarrier". (In object-oriented terminology, the "Car" class is a subclass of the "WheeledCarrier" class, "Airplane" is a subclass of "WheeledCarrier", and "WheeledCarrier" is a superclass of "Car" and of "Airplane". Of course, cars are different from airplanes and their full descriptions will reflect the differences, but because of the use of inheritance, redundant description of their common characteristics of having wheels and being able to carry passengers is avoided.

For further general background regarding object technology to facilitate a better understanding of the invention, reference should be made to "Object Oriented Technology—A Manager's Guide", by David A. Taylor, copyright 1990, Servio Corporation.

With the development of object-oriented programming, several "object models" were further refined and developed, which specified the manner in which one was to define objects and their external interfaces, such various object models providing the aforementioned characteristics of encapsulation and inheritance among others. At an appropriately high level, these various object models are very similar, examples of which are the System Object Model (SOM), Common Lisp Object System (CLOS), Smalltalk, and C++. In essence, these various object models are simply a body of rules answering the question of what an object is, each offering slightly different answers when examined at lower levels. For example, various object models differ in their language syntax, and in how encapsulation and inheritance work.

As a direct result of these differences, one problem presented by the availability of different object-oriented languages and object models was that problems in interlanguage operability appeared, e.g. object programs could not be written with a mixture of such languages, thereby adversely impacting one of the major promises of object technology, namely reusability of code. In an effort to address this problem plaguing the industry, a Common Object Request Broker Architecture (CORBA) was arrived at by committee, which included a standardized Interface Definition Language (IDL). There was in essence an agreement in the industry as to how interfaces of an object would be specified, i.e. a standard for defining object interfaces so that objects defined by one vendor could be utilized by another. Thus, with CORBA, the effort was started to facilitate uniform definitions across languages of what an object was to "look like" in order to facilitate implementation of applications in multiple languages.

The aforementioned System Object Model (SOM) is one object model which conforms to CORBA and IDL. Con-

formance means that SOM objects follow CORBA semantics, and SOM objects are defined in IDL syntax. The significance of the foregoing will become readily apparent hereinafter in a more detailed description of the invention but is provided at this point for background.

We will now turn more specifically to an application of this object technology, in particular, to problems associated therewith which have plagued the industry and been successfully addressed by the subject invention. One important use of object technology is in implementing an improved "visual builder" environment for visual programming. A visual builder is essentially an application writing tool permitting the "writing" of programs visually (instead of by the more traditional method of writing textual code). Particular utility for such tools arises with respect to the writing of programs having graphical-user-interfaces ("GUIs"). As will be hereinafter seen, the invention provides such a program-writing tool which is itself visually based as well.

It is a characteristic of current programming technique that a great deal of the time is spent in coding the actual graphical user interface or "GUI" presented to the user (as opposed to the underlying code and functionality to which the GUI interfaces), and the present invention is directed in part to simplifying these tasks. To be more clear as to what is meant by a "visual builder", put simplistically, it is a tool which facilitates the writing of programs by the creation, movement, and interconnection of icons, e.g., through a user interface or GUI as opposed to through conventional textual programming.

A simple example of such a visual builder may be seen in the product "Visual Basic" by Microsoft Corporation in which the user interfaces, such as various windows, pull-down menus, buttons, scroll bars, and the like are created visually. A visual builder system arranges for the actual functioning interface to appear when the built application is executed, thereby relieving the programmer of the task of writing a textual program to implement the interface. As will become more apparent hereinafter, there is an increasing need to facilitate the ability to efficiently program applications visually, e.g. by "visual programmers" who do not have the detailed knowledge of computer programming languages and the like associated with the more traditional notion of a computer programmer. This need has been fueled in part by the phenomenal increase in multimedia. The invention is intended to address in part the problems associated with this concept of visual programming.

A major task of implementing a useful visual builder is to provide the function of a visual editor of the tree or graph structure of FIG. **4** (although as will hereinafter be seen it will necessarily have additional requirements, such as providing the ability to edit properties associated with the objects, etc.).

In order to more fully understand the invention, again, by way of background, a general description of the requirements for such a visual builder will now be provided.

Referring first to FIG. **2**, a representative user interface **40** is shown which might appear on the monitor **38** of FIG. **1** during the process of a user visually programming a desired application. The purpose of the interface is to provide a dialog and interface between the computer and the programmer by means of visual items such as sliders, buttons, etc. whereby the programmer may proceed to structure, build, and define an end-user application. The application itself may also include graphical user interfaces employing items similar to those shown in the application build tool screen of FIG. **2**, e.g., sliders, buttons, windows and the like.

More specifically, a representative interface **40** might include various icons such as slider bars **42, 46** (with associated slider "thumbs", **43, 49**; end buttons **41, 45, 47, 48**); buttons **50, 58**; icons such as speaker icon **44**; windows **52, 56**; and pull-down menus **54**.

One aspect of the interface **40** of FIG. **2** not intuitively obvious is that although these various items appear "flat", "under the covers" they would reveal an associated nested hierarchy of visual objects such as depicted in FIGS. **3** and **4**. Thus, comparing FIGS. **2** and **3**, it will be seen for example that button **58**, rather than being merely a "flat" arrangement as a control, exhibits a nested relationship whereby button **58** is, hierarchically speaking, in box **56** which is in turn in box **52** and in turn in box **49**.

A conventional way in the art to represent a nesting or containment relationship, such as that embodied in a GUI and the corresponding programming code implementing it, is by means of a data structure commonly known as a "tree" or graph shown in FIG. **4**, which obviously more readily visually depicts the hierarchy. Each of the icons, slider bars, and the like are shown in FIG. **4** as a node or circle representing an "object" within the meaning of the object-oriented technology hereinbefore described.

In a conventional approach to writing user interface programs, trees similar to FIG. **4** are programmed textually. The various nodes, children to nodes, and children of children to nodes, etc. shown by the objects of FIG. **4** are typically implemented in building up the tree of FIG. **4** by coding in a high level language such as C, C++, or the like. As has already been pointed out, this process is extremely tedious, involving all of drawbacks of even high level languages including statements requiring precise arguments, syntax, semantics and the like.

Because of the foregoing, it became desirable to provide for the aforementioned "visual builders" whereby a program could be constructed which would in essence build the desired runtime trees of FIG. **4** by means of interactively creating, moving, and interconnecting various icons and the like (such as those depicted in FIG. **2**) inasmuch as it is far easier to do so by this means rather than by writing C programs, for example.

Thus, in response to this need, vendors set upon the task of providing visual builders to interactively build up these containment structures or trees.

It should be noted that interacting with the various items of FIG. **2** would not necessarily impact the runtime tree of FIG. **4**. For example, clicking on button **58** would not necessarily alter the tree structure of FIG. **4**. However, moving button **58** out of box **56** to location **60** would change the hierarchical structure of this object, as may be seen in FIG. **4**. Object **58A** which extended from object **56A** (corresponding to button **58** being contained within box **56**) would, after the button **58** was moved to location **60**, be reflected in the tree as object **60A** now extending from object **52A** (thereby reflecting the new hierarchy in which button **58** at location **60** is contained by box **52**). Thus, in general, it may be seen that manipulating the items in the interface FIG. **2** may alter the corresponding object tree of FIG. **4**.

A major task of implementing a useful visual builder is to provide the function of a visual editor of the tree or graph structure of FIG. **4** (although as will hereinafter be seen it will necessarily have additional requirements, such as providing the ability to edit properties associated with the objects, etc.).

Continuing with the general background discussion regarding the requirements of implementing visual builders

and user interfaces, textual programming of GUI applications, for example, with the C language and some library such as X and OSF/Motif, or the Presentation Manager library of the OS/2 operating system shows that several further tasks are required beyond the building of the initial tree. The initial properties of the various nodes in the tree must be set, or, in general, the programmer must decide, for each available property, whether to specify its initial value or to accept the default initial value. (Examples of properties of the nodes in the interface of FIG. **2** would be the color of box **49**, initial position of a slider thumb **43**, contents of a pull-down menu **54**, etc.). In addition to the task of setting or defaulting such properties of objects, there is the further task of making connections between the objects, thereby specifying the actions to be performed when certain events occur. For example, if the user were to depress button **50**, the desired action might be, for example, a beeping of speaker **44**. If the position of slider **43** changed, the desired action might be for some window to scroll appropriately.

Finally, after the tree has been built, the various object properties have been set, and the actions to be performed on specified events have been specified, the last task that a GUI application (either built by a visual builder or textually programmed) must do is to enter an event loop at runtime. If an event occurs, the program will perform the corresponding action, and then wait for occurrence of the next event. The typical desired behavior is for the application program to be running in an infinite event loop, responding to events such as the activation of button **50**, etc.

From the foregoing, it will be readily appreciated, in summary, that a user interface application may be represented by a tree such as that of FIG. **4** which in turn represents a hierarchy of objects, and that the construction of such a tree textually is an extremely tedious matter, giving rise to the development of visual builders. However, even with the presence of visual builders, several serious problems nevertheless persisted.

First, these objects themselves (such as those shown at the nodes in FIG. **4**), are typically written in accordance with more traditional programming techniques wherein the objects might never have been intended to be manipulated in a visual builder environment, but rather, intended to be part of a low level object-oriented toolkit library. Consequently, usability of these objects for higher level visual programmers may not have been a prime consideration or even a consideration at all in implementing these objects, and accordingly the properties thereof, prior to the invention, might not appear intuitive to a visual programmer.

A problem is that, in present systems, the interfaces of the runtime objects presented to the visual programmer are constrained to be those specified by the object definition. Thus present systems are limited to presenting to the visual programmer exactly the same interface (e.g., Smalltalk methods, CLOS generic functions, and by extension to CORBA, IDL attributes and operations specified by the definition of the object). Such interface may include, for example, unwieldy attribute or operation names, and attributes or operations which are at such a low level as to seriously impair the user friendliness of a visual builder.

The invention disclosed is our co-pending patent application Ser. No. 08/357,834, entitled "SYSTEM AND METHOD FOR PROVIDING A VISUAL APPLICATION BUILDER FRAMEWORK" filed on Dec. 16, 1994, solved these problems and provided a visual builder wherein the object's interface as seen by a builder user could be decoupled from that specified by the object model interface.

More particularly, it provided for an arbitrary mapping of more intuitive properties and operations of objects to the attributes and operations of runtime objects dictated by the particular object model. More control was provided in the manner in which editable properties and operations of runtime objects were presented to the visual builder user to avoid overwhelming the visual programming with unnecessary object detailing. The invention facilitated the definition of new properties of objects at higher levels of abstraction presented to the visual builder user, thereby making the editable runtime objects more intuitive.

The foregoing was accomplished by providing "proxy objects" and a system and method for visually constructing and editing a proxy object tree. Each such proxy object in the proxy object tree at build time corresponded to a correlative target object in a corresponding run time target object tree.

The proxy object interfaces, as seen by the visual builder, were "decoupled" from the interface specified by the particular object model of the target objects. In this manner, since the proxy object could be customized, the visual builder system was not forced to present to the user the exact attributes and operations of the target objects dictated by a given object model.

These proxy objects each required a great deal of information such as the class of the target object corresponding to the proxy object, presentation information on how the proxy object should present itself to the user, properties of the particular proxy object and how they are mapped to IDL attributes, and the like, all of which are further described herein.

A serious problem presented in implementing a visual builder employing the thus-described proxy object concept was how to efficiently and effectively store and access this large volume of proxy object information.

One approach was to hardcode in the implementation of the actual of the proxy itself all the necessary information for each proxy object.

Whereas this has the specific performance advantages typically associated with hardcoding, several serious disadvantages were nevertheless present as well, also associated with the nature of hardcoding.

One such disadvantage is that alteration in these proxy objects required altering the source code implementing the proxy object, which required a programmer well versed in the intricacies of writing program code.

Yet another significant disadvantage was that these changes to source code necessitated resultant recompilation of the source code.

SUMMARY OF THE INVENTION

It is therefore an object of the invention to provide a visual builder which is easy to maintain and alter.

It is a further object of the invention to provide such a visual builder which could be maintained without necessitating the alteration or rewriting of source code.

Yet a further object of the invention was to provide an improved visual builder based upon an object model implementation employing proxy objects, wherein modifications to such proxy objects was facilitated.

Still another object of the invention was to provide for such an object-based visual builder wherein modifications to such proxy objects could be effected easily by someone without detailed programming skills.

Still a further object of the invention was to provide for such a visual builder wherein modifications could be

effected to the proxy objects without necessitating recompilation of source code of such proxy objects.

An object model-based visual builder is provided which includes proxy objects at build time, each corresponding to a target object at runtime. Each such proxy object has associated therewith several items of information, including the class of the target object corresponding to the proxy object, presentation information, properties and how they are mapped to IDL attributes and operations, events available on the target object, and operations supported by the target object. A portion of such information is stored in an Interface Repository, such as the System Object Model (SOM) Interface Repository, in easily changeable form, with the necessary knowledge for retrieving such information from the Interface Repository being contained in the proxy object itself. The information stored in the Interface Repository may be changed without altering or recompiling the source code which implements the proxy object itself. Interface declarations for proxy objects are provided in corresponding IDL files stored in the Interface Repository. Implementation statements in the files permit modifiers which encode the easily editable portions of the proxy object information.

These and other objects have been provided by the invention, a more detailed description of which follows hereinafter and may be more fully understood with the accompanying drawings wherein:

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a block diagram of a personal computer system for use in implementing the subject invention;

FIG. **2** is an illustration of a graphic user interface portion of a program which may be implemented with the invention;

FIG. **3** is an illustration of components of the user interface of FIG. **2** illustrating their hierarchical nature;

FIG. **4** is an illustrative runtime target object hierarchy tree of the invention corresponding to FIG. **3**;

FIG. **5** is a create/edit/build proxy object tree in accordance with the invention corresponding to the tree of FIG. **4**.

FIG. **6** is a graph illustrating improvements in real memory usage in accordance with the invention;

FIG. **7** illustrates the storage of selected proxy object information in an Interface Repository;

FIG. **8** illustrates an example of a particular form of a proxy object and associated IDL file stored in an Interface Repository as illustrated in the FIG. **7**.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

From the background of the invention, the general benefits to a visual builder or interactive user interface development system should now be readily apparent. Before describing in more detail aspects of the subject invention, additional detail will be provided in relation to the requirements for a visual builder generally.

As previously described, the first task of such a builder is to perform the function of a tree editor and make building a tree such as that shown in FIG. **4** considerably easier than in the case of writing textual code, which would typically require code to create each node and attach it to the appropriate parent. With the advent of visual builder environments, the visual programmer may simply drag a desired object such as button **58** of FIG. **2** to location **60**. Internally, the visual builder code would automatically understand how to alter the corresponding object tree, thereby producing a new tree as shown in FIG. **4**.

As touched on in the Background of the Invention, a second task to be provided by a visual builder in accordance with the invention is to facilitate the setting of properties of the runtime objects such as those of FIG. **4**, i.e., the visual builder must be a property editor. This means that the builder must provide for the editing of initial properties of objects interactively without the necessity of the programmer writing code. For example, the programmer should be able to click on an object, such as one of the pushbuttons shown in FIG. **2**, and be shown a list of properties of that button object in order to set them. Such properties might include the foreground and background color, as an example.

A third function to be performed by a visual editor in accordance with the invention is to facilitate the user's ability to specify actions to be performed. Thus, a connection model will accordingly be provided, an example of which is as follows. The user might desire, for example, upon clicking on the button **50** of FIG. **2** with an appropriate pointing device, that a sound occur such as that represented by the speaker **44**. A desirable implementation of a system for specifying actions to be performed, using this example, would enable the user to drag a cursor after clicking on the button **50** to the desired object such as the speaker **44**. An arrow **62** might then be presented signifying that an operation will be performed on the target object **44** to be specified. Desirably a dialog box may then appear when editing this connection, shown by arrow **62**, to enable the visual programmer to specify the event on button **50** which would cause some specified operation on speaker **44** to be invoked. Possible events which could be associated with the button might be a "click" event, (occurring when the button is pressed), an "expose" event (occurring when the button first appears or when a window covering it is moved), or a "resize" event (occurring when the button is resized, if it is resizable).

As another example of objects having events for which the visual builder must be able to specify actions, consider slider object **42** in FIG. **2**. In the case of such a slider, a typical event might be a "changed" event if the position of the thumb **43** or **49** changed, thereby requiring the action of scrolling a window. In like manner, clicking by a pointer device on the end buttons **41**, **45**, **47**, or **48** might, in response to this "click" event, cause scrolling by a page in a window.

From the foregoing, it will be seen that a way may thus be provided for interactively specifying actions to be performed when an event occurs. In summary, then, a desired implementation of a visual builder in accordance with the invention builds an object tree associated with an interface, facilitates setting the properties of the objects, and then further facilitates specifying the actions to be performed when certain events occur.

Detail will now be provided of how a particular visual builder framework in accordance with the present invention operates.

It will be noted that FIG. **4** is labelled a "Runtime Target Object Hierarchy Tree". Nodes shown in the Figure such as **72**, are representations of actual objects which are desired to be created when the end-user application executes. At realization, an application built with the visual builder of the invention will, at runtime, create such a "runtime" object tree of FIG. **4**. Secondly, it will be recalled that the various properties of the nodes or target objects shown in FIG. **4** will

be set, and finally all of the linkages such as those shown by arrows **84** will be effected (specifying that if an event occurs on a given object, this will in turn call an operation of another (possibly several) object). FIG. **4** is thus labelled "Runtime" because it illustrates the objects which the application must create when it runs.

It is important to make a distinction between this runtime and "build time" during which the visual builder is not actually creating the objects of FIG. **4**. It is an important feature of the invention that "proxy" objects are created by the visual builder during this "build time" (shown in FIG. **5**), and that these editable proxy "source" objects map to corresponding runtime "target" objects shown in FIG. **4**.

By analogy, in standard textual programming, at "build time", when an end-user application is being written, the programmer is editing source code text which is not (usually) the actual executable form of the program. Thus, in the case of the visual builder of the invention, the visual builder, in part, may be thought of as a "source code" editor performing a function similar to the text-based source code editor in the more conventional process of editing in standard textual programming. The distinction is that the "source" being edited by the visual builder is not a stream of ASCII text characters, but rather a tree of these "proxy" objects, as shown in FIG. **5**.

Whereas at runtime (FIG. **4**), the goal is for the visual builder to build the tree of "target objects", at build time (FIG. **5**) it is desirable for the visual builder to create, edit, and build a tree of corresponding proxy objects. A comparison of FIGS. **4** and **5** will reveal a similar pattern between the target objects and proxy objects visually illustrating their one-to-one correspondence and the similarity in structure between the proxy object tree and the target object tree.

Thus, the visual builder of the invention may be seen in reality to be, in part, a proxy tree or graph editor, with the proxy objects being the "source" which is manipulated and edited by the visual programmer at build time. The reason why a proxy tree of FIG. **5** is required is that the proxy objects differ from the target objects in important respects which will be hereinafter detailed.

As previously described, each of the target objects shown in FIG. **4** (**70**, **72**, etc.) represent the final actual runtime objects which are desired to be built, such as the scroll bar, window, form, etc., objects. Although the examples discussed have focused on visual runtime objects, note that runtime objects may also be nonvisual, e.g., they may not necessarily have a visible representation at runtime. In contrast, the objects represented by the circles or nodes in FIG. **5** are "stand-in" or "proxy" objects (which may also be thought of variously as "surrogate" or "shadow" objects) in the sense that each one has a corresponding target object at runtime. This explains the numbering convention of "**70**" for the target object and "**70A**" for the proxy object, as an example.

Now that the notion of a "proxy" object has been introduced as a fundamental concept of the present invention, it is necessary to detail the functions it must be capable of performing. They are as follows:

Representative Proxy Functions

1. Present itself at build time.
2. Provide storage for, and access to, property values.
3. Provide information on the events available on corresponding target object.
4. Provide information on the operations available on corresponding target object.
5. Save itself on, and restore itself from, a persistent medium.
6. Copy itself.
7. Realize itself, and subsequently detach itself from the realized target object tree, and destroy itself.

As to the first function of all such proxy objects, e.g., presenting itself at build time, this means that each proxy object must be able to display on the appropriate monitor **38** of FIG. **1**, a visual manifestation of itself to enable interactive manipulation. Thus, as an illustration, a function of the proxy object for a button **58** of FIG. **2** must be to present a visual image in a user interface so as to suggest to the visual programmer that there exists a button target object in the target object tree being built.

Regarding function **2** of proxy objects, it will be recalled from the foregoing that a visual builder of the invention serves as a property editor, i.e., it provides the facility for editing the various properties of proxy objects. Accordingly, it should be evident that it is necessary to provide some facility for keeping track of these properties, e.g., what names and current values are. As an example, a pushbutton target object and corresponding proxy object might have, as properties, the foreground and background colors thereof, height, width of the button, and the position of the upper-left corner thereof. A corresponding proxy object would therefore store this property information associated with the target button object. In other words, the button proxy object, as an example, will "know", (i.e., be capable of storing and accessing) the aforementioned properties and what their current values.

Regarding functions **3** and **4** of the proxy objects, if the visual builder of the invention is going to include the facility of making interconnections between proxy objects, there must be facility for knowing the events and operations which are available. For example, if proxy object **74A** corresponds to a pushbutton target object **74**, then an additional function of the proxy object **74A** is to have information on the events which can occur on it, such as the "click" event. In other words, the proxy object must know the events available so that the programmer may select the appropriate one to which to attach a desired action. Actions are usually invocations of operations on other objects. Therefore, a proxy object must have information on the operations supported by its corresponding target object so that the programmer may select and properly call the appropriate one in response to an event. For example, if it is desired to call an operation on the "speaker" object, then the speaker proxy object should know that its corresponding target object supports a "beep" operation.

Regarding function **5** of proxy objects, in the course of visual programming of an application, it may be desirable to suspend creation of the program. This proxy object function simply means that each proxy object must be able to save its corresponding proxy tree information and associated data. In other words, proxy objects must be provided with the ability to save themselves to, and later restore themselves from, some persistent medium such as a file system when visual programming of the particular application is resumed. This function is analogous to the ability of most text editors to save and later resume editing of textual source code.

Regarding function **6** of proxy objects, such proxy objects must be capable of replicating themselves. This facility is required for convenience in visual programming, i.e. to make it possible to create copies of objects at build time.

Finally, function **7** of proxy objects means that proxy objects be capable of realizing themselves, and subsequently detaching themselves from the realized target object tree,

and destroying themselves. Proxy objects may be thought of as "delegates" which essentially are standing-in for their corresponding target objects. At some point, it will be desired to execute the program being built, and it will therefore be necessary to create and initialize the runtime tree, such as the example tree shown in FIG. 4. The collection of proxy objects has the necessary information to effect the running of the program inasmuch as it is essentially the "blueprint" or "source code" (FIG. 5) for the corresponding application (FIG. 4). The term "realize" simply means that the proxy objects must be capable of creating their associated target objects.

There are numerous benefits to providing a visual builder of proxy objects and the proxy object tree of FIG. 5 which may not be readily apparent and will thus be hereinafter described in greater detail. The description of these benefits will, in turn, further dictate the manner in which such proxy objects and the corresponding visual builder will be implemented in accordance with the invention.

A first major benefit of the visual builder implementation currently being described, which provides a significant distinction over visual builders of the prior art, is that by provision of such proxy objects, the proxy objects' interfaces as seen by the visual builder user are thereby decoupled from the interface specified by the target object definition.

The target objects of FIG. 4 are defined in a particular object model such as SOM, in the case of the implementation of the invention being presently described. As such, these target objects have an IDL definition which define what the attributes and operations of the particular target object are.

Because, in constructing the visual builder, it has been implemented so that the various proxy objects may be pre-defined and customized, the invention's system is not forced to present to the user of the visual builder the exact attributes and operations of the target objects of FIG. 4. Rather, the flexibility to define a different set is provided.

An example of the foregoing will be helpful in understanding its significance. Operations associated with a target object such as object 72 of FIG. 4, as dictated by the IDL, may be extremely low level. In like manner, attributes of such target objects 72 may have long and rather difficult names to read. Because the invention provides the facility for proxy objects, a different set of properties may be presented to the visual programmer which map to corresponding attributes on the target object. As an example, let us suppose that target object 72 has these attributes, namely "red", "green", and "blue", which are components of an RGB specification of the color of some part of the object on the screen. (Each component controls the amount of the associated primary color.)

At the visual builder level, it is undesirable to present to the visual programmer the information that this particular target object has three properties of "red", "green", and "blue". It would be much more desirable to simply communicate to the programmer that object 72 has a "color" property whose values may take on those listed such as pink, purple, orange, etc. These colors would then be translated into the necessary R, G, B components which the IDL attributes provide in order to effect the desired color.

In other words, a given target object might not provide a "color" attribute in its interface definition. Rather, it may have three separate attributes which collectively make up the color (e.g., R, G. B). To a visual programmer, this may appear as an unduly low level way in which he or she is required to specify color. It would be much more preferable to only require the visual programmer to pick the particular desired color such as orange, rather than doing so by way of specifying 100% red, 50% green, and 0% blue.

Thus, in providing for proxy objects, the invention allows a single property like "color" to be defined and presented to the programmer, and the proxy object possesses the information to translate this user-specified "orange" color value into the attributes of the target object, namely 100% red, 50% green, and 0% blue. For relatively high level users, the invention has facilitated the ability to define new object properties which will then be translated (transparently to the user) into the necessary target attributes. The attributes of the target objects can therefore be rendered more intuitive to the visual programmer by means of such proxy objects.

Stated another way, proxy objects provide a mechanism for introducing abstraction in an object's properties and for mapping the abstractions onto the actual defined interface. If we thus were to define a button's "color" property in the corresponding button proxy object, this proxy object will contain the information necessary to know how to set the corresponding appropriate R, G, and B IDL attributes in the target object the high level "color" property. As will be hereinafter made clearer, the knowledge of how to map the high level proxy object properties to the target object attributes will be utilized at build time and at realization time, and the knowledge is built into the particular proxy object when it is designed.

The invention need not be limited to only mapping properties of proxy objects to one or more attributes (such as IDL attributes in the implementation being described). The invention specifically contemplates mapping properties of proxy objects additionally to arbitrary combinations of properties and operations.

As an example, continuing with the prior illustration, it may be desirable to specify a "color" with R, G and B components. A target object may not have not three IDL attributes R, G, B, but instead may have three IDL operations such as "setR", "setG", and "setB". We may further desire that the builder user again simply be required to set a property "color". In this case, the proxy object would contain the necessary information to translate this "color" into calls to the three operations "setR", "setG", and "setB". This is thus an example of a proxy object property such as "color", being mapped to several operations, rather than an attribute. Note that an "operation" on an object is something which is performed or invoked on an object, as distinct from an "attribute", which is data which may be set or queried, such as "color" or "speed". However, in most object models, including SOM, attributes are just a special case of operations.

From the foregoing, it should be clear that providing the ability to define proxy objects enables us to change the way an object looks to a user, and to not be constrained by the available interface of the runtime target object.

We could, in designing a proxy object for a particular target object, add, remove, or redefine properties of the target object, as long as the new set of properties can be mapped to the existing available attributes and operations.

There is yet another important benefit to having proxy objects. Specifically, provision for proxy objects facilitates the implementation of visibility or "exposure" control on properties and operations of proxy objects.

Typically a visual programmer may not desire to be overwhelmed with the need to view all available editable properties or callable operations of a runtime target object. Provision for proxy objects enables us in the design of the visual builder to present these properties or operations to a

visual programmer in a more organized manner. They may be classified into related groups, and a user may be required to take an affirmative step such as clicking on a button to see the properties in a given group. This is what is meant by visibility or exposure control. The user may initially only be shown those properties which are most typically desired to be edited or selected. The proxy objects provide the mechanism to control what properties are seen and editable by a visual builder user at any time.

A point of clarification regarding terminology may be appropriate in order to avoid confusion. At some appropriately high level, the terms "property", "attribute", "characteristic", and "feature" are synonymous and may be used interchangeably. However, as a matter of convention, we use the terms "attribute" and "operation" when referring to the IDL-defined interface of a target object, and the terms "property" and "operation" when referring to the proxy-defined interface, as seen by a visual programmer.

As was explained in the Background of the Invention, if the implementer of a visual builder could control the design of the target objects and their specifications, there might not be a need for redefining the set of properties seen by a visual programmer, or for controlling their visibility. However, in practice, such ability is not present because typically the target objects are already pre-written, often by another programmer under the constraints of a different usage model.

The proxy object framework affords the visual builder implementer (or perhaps more correctly, the person enabling a given object to be used and manipulated within the visual builder), the flexibility to essentially define a new "world" to suit desired purposes and needs. Typically, one of the most important of these needs is that the visual builder be user friendly and, secondarily, exhibit desirable performance characteristics. This flexibility is provided so long as we have the facility for translating the proxy object world to the target object world. The properties of particular proxy objects which are visible to a builder user do not have to be identical to the target object IDL attributes. On the contrary, we thus have the flexibility to define a different set of properties for purposes of user friendliness, for example, so long as we can map the "virtual" properties presented by the proxy objects to the target properties.

It will be noted that the invention is not limited to the mapping of properties. We may also redefine the set of object operations as seen from within the visual builder environment. The operations seen by the visual programmer need not be identical to the operations supported by the corresponding target object. As in the case with proxy object properties, we can relax the requirement that all target object operations be shown as proxy object operations. In other words, the proxy objects may advertise a different set of operations, again as long as we provide facility for translating the new operations to the attributes and operations of the target objects. The earlier comments on visibility control of properties also apply to the operations seen by the visual programmer when specifying event-action connections between objects.

Another important aspect and advantage provided by the invention, in implementing the visual builder and proxy framework as herein described, is the detachment and destruction of the proxy tree after it has performed realization at built application runtime, thereby saving memory consumption.

To explain this feature, recall that important functions of proxy objects is that they must be capable of "realizing" themselves, i.e. creating their corresponding runtime target objects. AT application runtime, the proxy tree of FIG. **5** facilitates the building of the corresponding target object tree of FIG. **4**, sets the properties of the target objects based upon the proxy object properties (since proxy objects contain all of the mapping knowledge necessary to set the attributes and call the appropriate operations on the target objects), and establishes the connections indicating, for example, that when event "E" occurs on object "X", operation "P" is to be called on object "Y", etc.

At this point, the necessary runtime object tree of FIG. **4** now exists and is read to "go", i.e., control can now enter the event loop previously described. Because the proxy tree (which, it may be recalled, may be thought of as "source" code) has performed its job and created its image, there is no more need for it to be consuming memory. Accordingly, it is a further feature of the invention to provide for the detaching and destroying of the proxy objects in real memory after they have effected the realization, at runtime, of the target object tree of FIG. **4**.

Referring now to FIG. **6**, during the first few moments of application runtime execution, the proxy tree (FIG. **5**), is created in real memory and is needed in order to build the runtime tree of FIG. **4**. However, after all target objects are built, and the appropriate attributes are set and operations called to initialize the objects' state, and the event-action connections are established, the proxy objects may be removed from memory. The graph **90** represents real memory usage of a system such as that of FIG. **1** wherein the invention is implemented. An initial spike **96** shown therein is caused by the loading of dynamic link libraries (DLLs) associated with the proxy objects into real memory and the restoration of the proxy tree at the beginning of application execution. These DLLs contain the code associated with methods of all the proxy objects. The dip after the spike at point **94** suggests the significant drop in memory usage corresponding to the destruction of the proxy object tree of FIG. **5** and the unloading of such DLLs of the proxy object tree after construction of the target object tree. In other words, the spike **96** is caused in large part by the DLLs of proxy-related code loaded in real memory as well as the proxy object tree. Upon unloading of the DLLs and destruction of the data structures of the proxy tree, which is no longer required, real memory usage drops significantly as shown at reference numeral **94**.

It will be recalled that seven functions were enumerated which must be performed by proxy objects. A related consideration is what information these proxy objects must contain to carry out their functions The information may be categorized as follows:

1. The class of the target object corresponding to the proxy object.
2. Presentation information.
3. Properties and how they are mapped to IDL attributes and operations.
4. Events available on the target object.
5. Operations supported by the target object.

More specific detail will now be provided as to these items of information which proxy objects must contain.

As to the class of the target object, if, as an example, a proxy object is a representative or "proxy" for a "President" object (i.e., the class of the target object is the "President" class), the proxy object, at realization, must "know" that it must create a "president" target object corresponding to itself (the "president" proxy object).

As to the second category of information which the proxy object must contain, recall that one function of the proxy

object is to present itself (i.e., display something on the screen) at build time. Therefore, a proxy object must contain the information necessary to effect this presentation. The proxy object must know what object to display to the user, and if, for example, the user has iconified an object, what icon to display.

As to the third item of information contained by proxy objects, e.g., "properties", this refers to the fact that proxy objects must know the properties which can be edited since, at build time, the visual builder user will be editing such properties. The proxy object needs to know the name of each property, the data type of property (e.g., integer, Boolean, string, etc.), the current value, and information relating to visibility control. It will be recalled that it is a feature that not all properties need be exposed to the user, and that differing levels of property or operation disclosure or grouping may be implemented. The proxy object must also include knowledge on how the properties are mapped to the IDL attributes and operations.

Fourth, the proxy object must contain information regarding events available on the target object. Using the "President" example, if a proxy object represents a "President" target object, the proxy object must know the available "events" of the target object. For example, the proxy object must know that the "President" target object has a "speak" event available.

Finally, proxy objects must contain information indicating the operations the target object supports including information on the operation name, arguments, and return values, and how they are mapped to IDL attributes and operations.

One problem in implementing a visual builder, given the need of the proxy objects to contain the aforementioned information, is how to efficiently and effectively store and access this information.

One approach is to hardcode in the (class of the) proxy object all information listed above for each class of proxy object.

An advantage to hardcoding this proxy object information is the performance benefit which is typically associated with hardcoding. However, a significant disadvantage is the lack of flexibility also typically associated with hardcoding, namely, that changes require altering the source code and recompiling, with all of the attendant drawbacks. As an example, if in the "President" proxy object it was hardcoded that the only available event was "speak", and later it was desired to add the event "eat", this would necessitate changing and recompiling the source code of the President proxy object.

Accordingly, it a feature of the invention to provide a list of events stored elsewhere and merely looked up by the proxy object. Instead of a proxy object hardcoding all of the information hereinabove listed, it is sufficient for a proxy object to know how and from where to retrieve the information. By doing so, there is no requirement to alter the proxy object code, but rather to simply alter the list of events, which is maintained separately from the proxy object code.

In general, the approach adopted by the invention is one in which some or all of the information required for a proxy object is embedded in the IDL definition of the (class of the) proxy object, which definition then appears in, and is accessible from, an Interface Repository to be hereinafter described. The knowledge of how to retrieve this information from the Interface Repository is hardcoded (in the proxy object class object).

FIG. **7**, provides an illustration of the foregoing concept. An example proxy object tree such as that in FIG. **5** is shown. Also shown is a SOM Interface Repository **100**. The list of events available on a particular target object is stored in the Interface Repository **100**. The proxy object **76** A for a particular target object knows how to retrieve this list from the Interface Repository **100**, rather than storing this information directly. In this manner, should it be necessary to alter the list of events, this may be done by simply editing the portion **106** of the Interface Repository **100** containing the list, thereby avoiding the need to recompile the proxy object code.

In general, rather than hardcoding all of the information **108** required by the proxy objects as shown by the legend **102** for each proxy object, part of this information **108** is stored as desired elsewhere in an Interface Repository **100** which may be more easily changed without the need to recompile. The information or knowledge required by the various proxy objects to locate the information in the repository **100** is what is hardcoded into the proxy objects. It will be noted that repository **100** is simply an information store which may be implemented in several ways, such as through a file system.

Considerations well known in the art will determine, for a given implementation, which of the hereinbefore listed items of information required by proxy objects will be hardcoded in the proxy objects and which will preferably be stored in the Interface Repository **100**. In the specific implementation being described herein, all items of information 1–5 shown at reference numeral **108** in FIG. **7** would be provided in the repository **100** with the exception of mapping knowledge properties and operations. At one extreme, all such information could be hardcoded into the proxy object, and at the other extreme, a great deal of information could be stored in the Interface Repository **100**. In the latter case, this may necessitate development of a language processor or parser. Thus, it may be readily seen that tradeoffs well known in the art present themselves regarding the decision of hardcoding versus development of sophisticated language processors. Put simplistically, in the implementation under discussion, items of information for proxy objects were placed in the Interface Repository **100** which could be represented as lists which could be easily processed, whereas other items which could be more easily hardcoded were placed in the proxy objects.

It should be pointed out that although storage in the Interface Repository **100** provides the advantage of avoidance of recompilation, for example, one disadvantage of storing a large amount of proxy object information relates to performance, e.g., I/O limitations to bulk storage devices such as DASD or the like, which would be a conventional way to implement the Interface Repository **100**.

More detail will now be provided regarding specifics of how the appropriate proxy object information is stored in the Interface Repository, with a specific instance of the technique in accordance with the invention discussed with reference to FIG. **8**. The technique of the invention employs SOM IDL modifiers which are the mechanism used to store information in the repository **100**. Such a modifier is another syntax in SOM class definition.

FIG. **8**, depicts a representative IDL file **112**, wherein a "President_Proxy" object class **110** is defined. In this example, the interface definition is in accordance with the IDL syntax previously described. Shown in the file **112** are the attributes and operations **114** of "President Proxy" objects. It will be recalled that defining an object in IDL syntax requires such attributes and operations **114** as mandated by CORBA.

It will be recalled that while SOM is IDL compliant, it goes beyond the requirements of IDL and CORBA in

defining additional semantics and syntax. SOM provides for an implementation statement **118** within an interface declaration, which in turn can include arbitrary modifiers **120** such as the "events" and "operations" modifiers shown therein. The basic idea and rationale for such modifiers **120** in SOM is that it is not possible to foresee all possible things which may be desired to be declared about a class. Thus, rather than restricting SON IDL syntax to include only a limited set of forms, an "escape valve" was provided in the form of these modifiers **120** which are ignored by the SON compiler. However, the modifiers **120** are stored in the SON Interface Repository **100** by the SON compiler, and thereby made available for access by any program, even though the SON compiler itself does not know how to utilize the information.

It should be apparent that "lists" of proxy object information which are desired to be stored in the Interface Repository **100**, in light of the discussion of modifiers, may be placed in an IDL file **112** as modifiers **120**. The SON compiler, as part of the compilation process, places interface definitions (including the modifier information **120**) in the Interface Repository **100**, so as to be available to any program knowing how to access the repository **100** (which, of course, would include code effecting the access and retrieval by the appropriate proxy objects of the object's information).

While the invention has been shown and described with reference to particular embodiments thereof, it will be understood by those skilled in the art that the foregoing and other changes in form and detail may be made therein without departing from the spirit and scope of the invention.

What is claimed is:

1. A computerized method for supporting a proxy object comprising:

storing a first portion of information corresponding to said proxy object in said proxy object; and

storing a second portion of said information comprising visual programmable interface information viewable at build time for supporting visual programmability of said proxy object at said build time, said second portion at least partially defining said proxy object in an interface repository; and

executing a visual builder function invoking said second portion of said information at said build time.

2. The method of claim **1** wherein said first portion enables retrieval of said second portion from said interface repository during said build time.

3. The method of claim **2** wherein said second portion of said information is imbedded in an IDL modifier of the class of said proxy object.

4. The method of claim **1** wherein said interface repository is a system object model interface repository.

5. The method of claim **4** wherein said first portion of said information is hardcoded.

6. The method of claim **5** wherein said second portion of said information is editable text.

7. The method of claim **6** wherein said first portion defines a metaclass of said proxy object.

8. The method of claim **7** wherein said second portion of information is selected from a group comprising properties of said proxy object and operations of said proxy object.

9. The method of claim **3** wherein said IDL modifier is a system object model IDL modifier.

10. The method of claim **1** wherein said first portion of said information at least partially defines said proxy object.

11. The method of claim **1** wherein said second portion of said information corresponds to a window object.

12. The method of claim **11** wherein said second portion of said information is size and position of said window object.

13. The method of claim **11** including:

generating a user interface to a programmer in response to said executing said visual builder function for facilitating programming of said proxy object at said build time.

14. The method of claim **13** wherein said programming comprises:

manipulation of said proxy object's runtime behavior.

15. The method of claim **14** wherein said visual builder function includes copy, save, and restore functions of said proxy object at said programming time.

16. The method of claim **15** wherein said proxy object is persistent.

17. A computerized system for supporting a proxy object in a visual builder function comprising:

means for storing a first portion of information corresponding to said proxy object in said proxy object;

means for storing a second portion of said information comprising visual programmable interface information viewable at build time for supporting visual programmability of said proxy object at said build time, said second portion at least partially defining said proxy object in an interface repository; and

means for executing a visual builder function invoking said second portion of said information at said build time.

18. The system claim **17** wherein said first portion enables means for retrieval of said second portion from said interface repository during said build time.

19. The system of claim **18** wherein said second portion of said information is imbedded in an IDL modifier of the class of said proxy object.

20. The system of claim **17** wherein said first portion of said information at least partially defines said proxy object.

21. The system of claim **19** wherein said IDL modifier is a system object model IDL modifier.

22. The system of claim **17** wherein said interface repository is a system object model interface repository.

23. The system of claim **22** wherein said first portion of said information is hardcoded.

24. The system of claim **23** wherein said second portion of said information is editable text.

25. The system of claim **24** wherein said first portion defines a metaclass of said proxy object.

26. The system of claim **25** wherein said second portion of information is selected from a group comprising properties of said proxy object and operations of said proxy object.

27. The apparatus of claim **17** wherein said second portion of said information corresponds to a window object.

28. The apparatus of claim **17** wherein said second portion of said information is size and position of said window object.

29. The apparatus of claim **28** including:

means for generating a user interface to a programmer in response to said means for executing said visual builder function for facilitating said programmer's programming of said proxy object at said build time.

30. The apparatus of claim **29** wherein said means for generating includes:

means for manipulation of said proxy object's runtime behavior.

31. The apparatus of claim **30** wherein said visual builder function includes means for copying, saving, and restoring functions of said proxy object at said build time.

32. The apparatus of claim **31** wherein said proxy object is persistent.

* * * * *

Modify (col 14), default (col 24), Compare (col. 20)

US005327529A

United States Patent [19]

Fults et al.

[11] Patent Number: 5,327,529

[45] Date f Patent: Jul. 5, 1994

[54] **PROCESS OF DESIGNING USER'S INTERFACES FOR APPLICATION PROGRAMS**

[75] Inventors: **Douglas A. Fults**, San Leandro; **Anthony M. Requist**, Alameda, both of Calif.

[73] Assignee: **Geoworks**, Berkeley, Calif.

[21] Appl. No.: **942,354**

[22] Filed: **Sep. 9, 1992**

Related U.S. Application Data

[63] Continuation of Ser. No. 681,079, Apr. 5, 1991, abandoned, which is a continuation-in-part of Ser. No. 586,861, Sep. 24, 1990, abandoned.

[51] **Int. Cl.**[5] **G06F 3/14**; G06F 9/45

[52] **U.S. Cl.** **395/155**; 395/700

[58] **Field of Search** ..... 395/155, 160, 156, 159, 395/157, 700 MS, 650 MS

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,692,858	9/1987	Redford et al.	395/157
4,782,463	11/1988	Sanders et al.	395/70
4,811,240	3/1989	Ballou et al.	395/155
4,866,638	9/1989	Cosentino et al.	395/159
5,179,657	1/1993	Dykstal et al.	395/155 X
5,021,976	6/1991	Wexelblat et al.	395/159
5,041,992	8/1991	Cunningham et al.	395/155 X
5,115,501	5/1992	Kerr	395/700 X
5,119,475	6/1992	Smith et al.	395/156
5,121,477	6/1992	Koopmans et al.	395/156
5,179,700	1/1993	Aihara et al.	395/650

OTHER PUBLICATIONS

"Making the Same Look Different"-ORACLE 1990 (Advertisement).

"Automatic, Look-And-Feel Independent Dialog Creation For Graphical User Interfaces", Brad Vander Zanden and Brad A. Myers; School of Computer Science, Carnegie Mellon University, CHI '90 Proceedings-Apr. 1990, pp. 27-34.

"Neuron Data Open Interface" Technical Overview The Tool for Building Portable Graphical User Interfaces Across All Windowing Standards-Neuron Data Inc., May 1991.

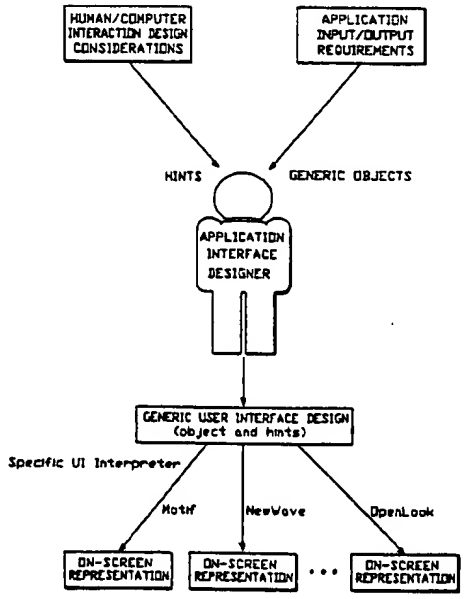
Primary Examiner—Raymond J. Bayerl

Attorney, Agent, or Firm—Wilson, Sonsini, Goodrich & Rosati

[57] **ABSTRACT**

A method for invoking a user interface for use with an application operating in a computer system which involves providing in the computer system a generic object class that corresponds to a class of function that is to be performed using the user interface; specifying in the application instance data in the form of a generic object specification that corresponds to the generic object class, the instance data including attribute criteria and hint criteria; providing in the computer system at least one specific user interface toolbox and controller that operates in the computer system to provide a selection of possible specific user interface implementations for use in performing the class of function; and providing in the computer system at least one interpreter that corresponds to the at least one specific user interface toolbox and controller.

11 Claims, 24 Drawing Sheets



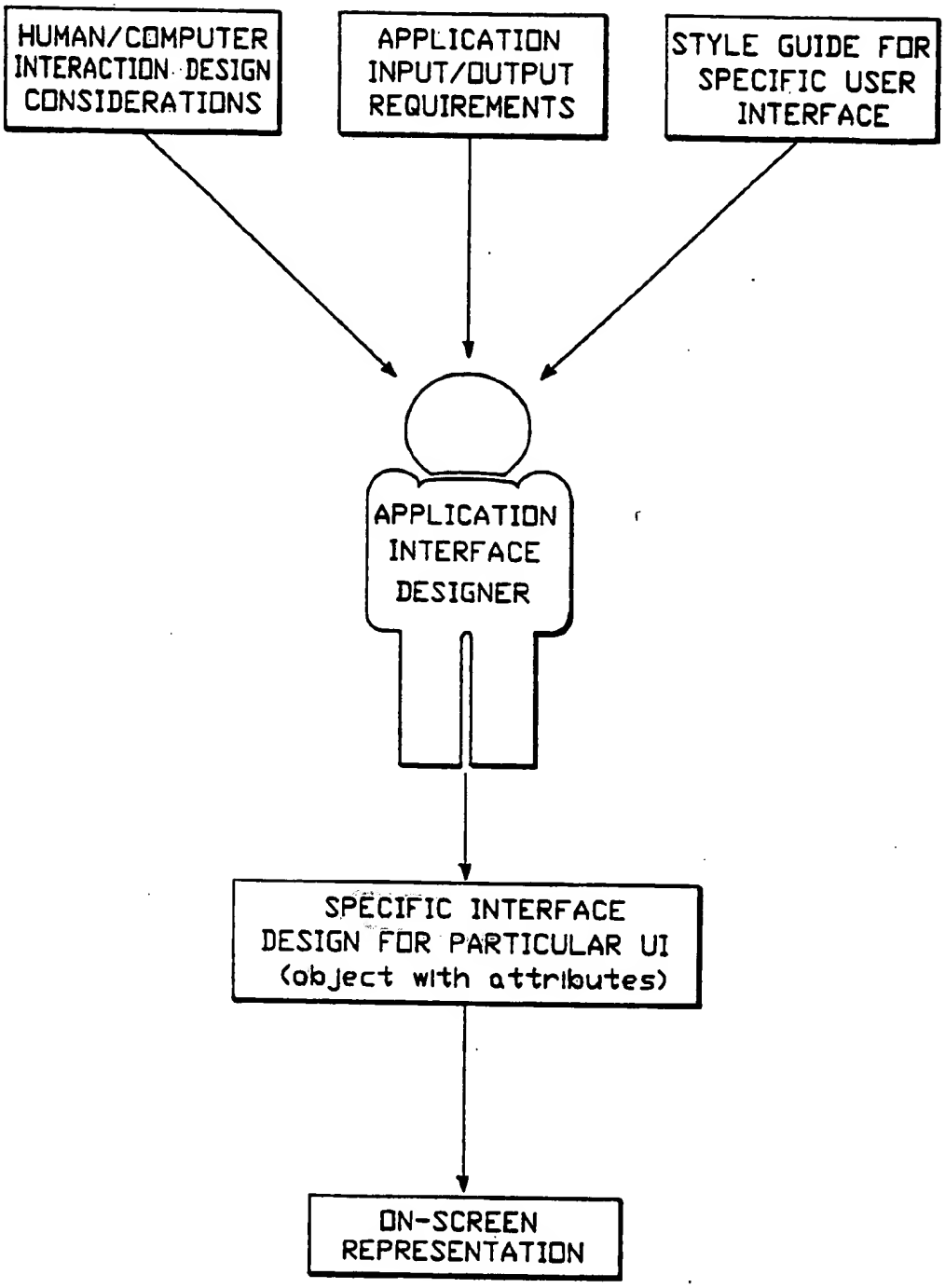

HUMAN/COMPUTER INTERACTION DESIGN CONSIDERATIONS
APPLICATION INPUT/OUTPUT REQUIREMENTS
STYLE GUIDE FOR SPECIFIC USER INTERFACE
APPLICATION INTERFACE DESIGNER
SPECIFIC INTERFACE DESIGN FOR PARTICULAR UI (object with attributes)
ON-SCREEN REPRESENTATION

FIG.—1

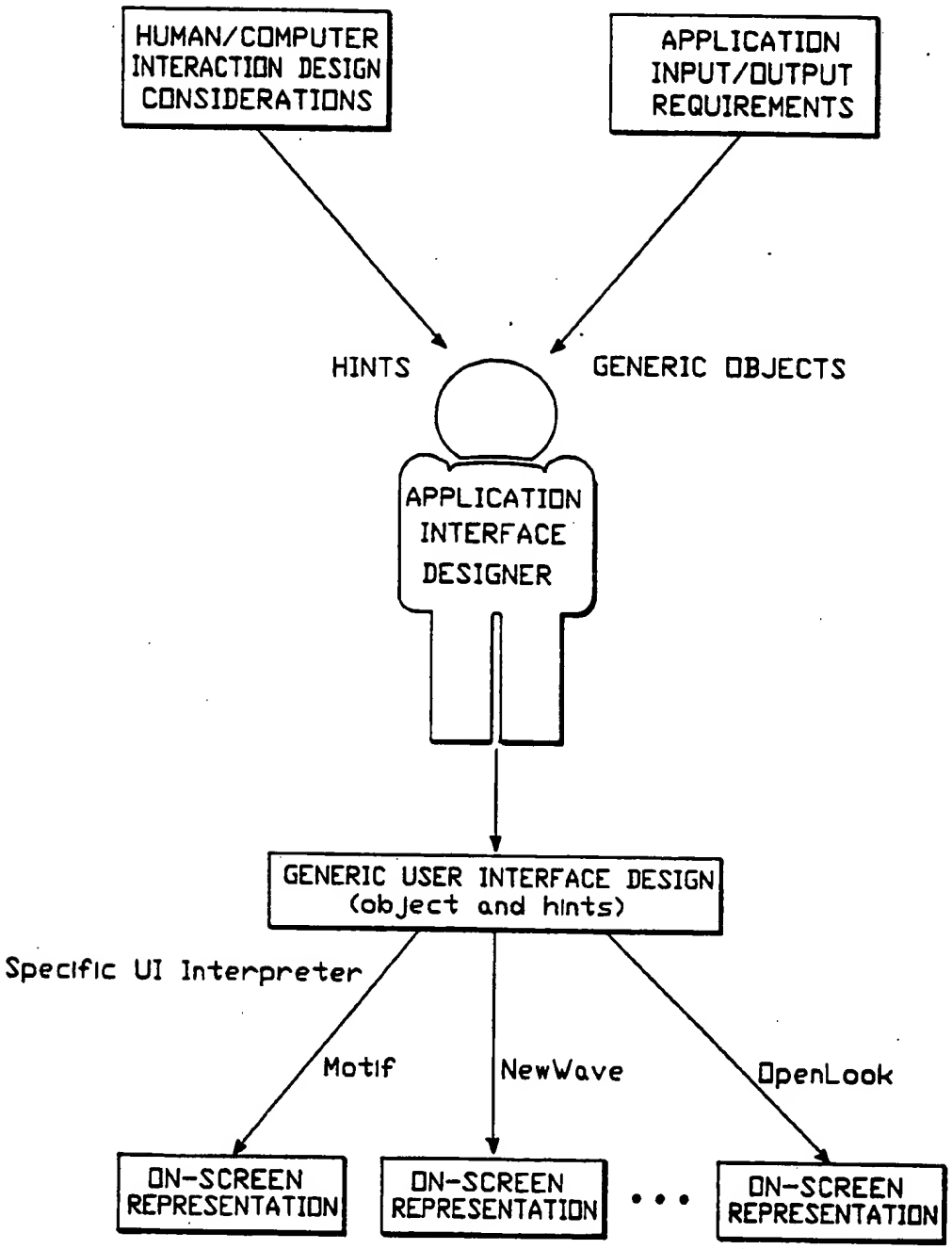

HUMAN/COMPUTER INTERACTION DESIGN CONSIDERATIONS
APPLICATION INPUT/OUTPUT REQUIREMENTS
HINTS
GENERIC OBJECTS
APPLICATION INTERFACE DESIGNER
GENERIC USER INTERFACE DESIGN (object and hints)
Specific UI Interpreter
Motif
NewWave
OpenLook
ON-SCREEN REPRESENTATION
ON-SCREEN REPRESENTATION
ON-SCREEN REPRESENTATION

FIG.—2

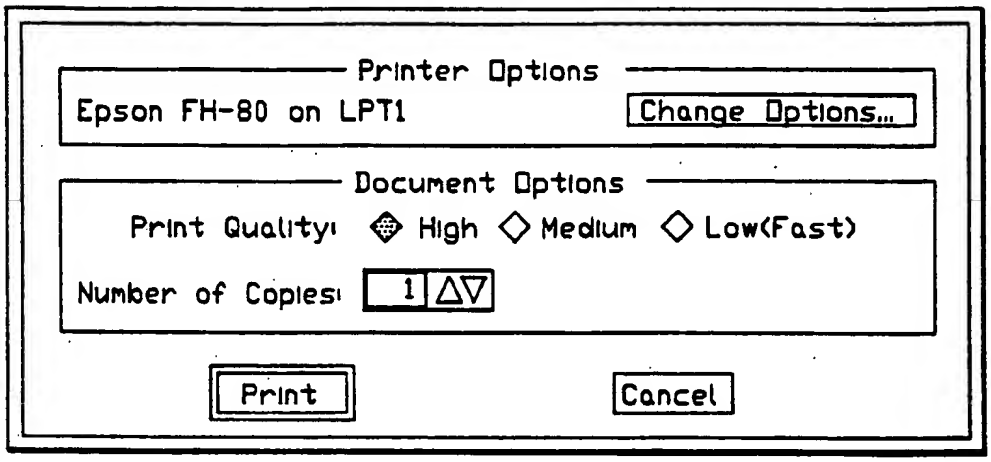


FIG.—3

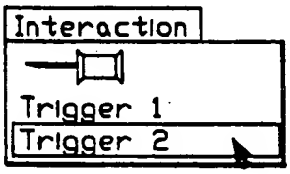


FIG.—4

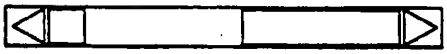


FIG.—6

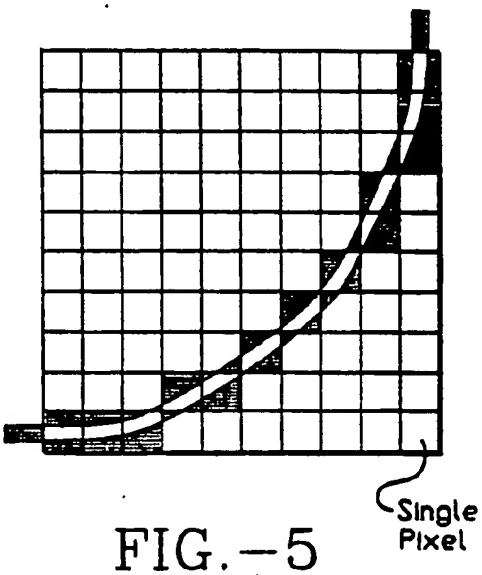


FIG.—5

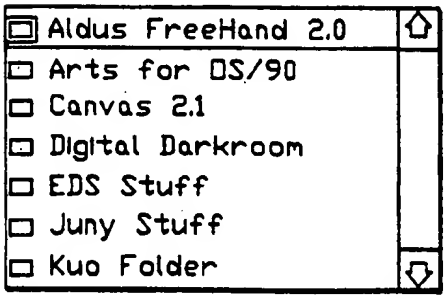


FIG.—7

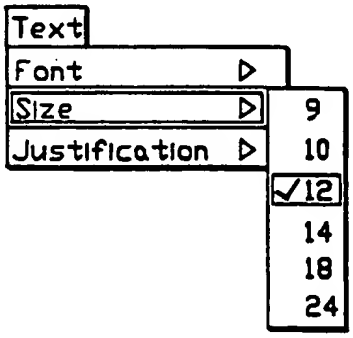


FIG.—8

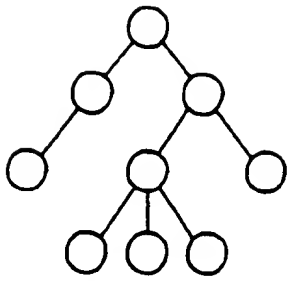


FIG.—9

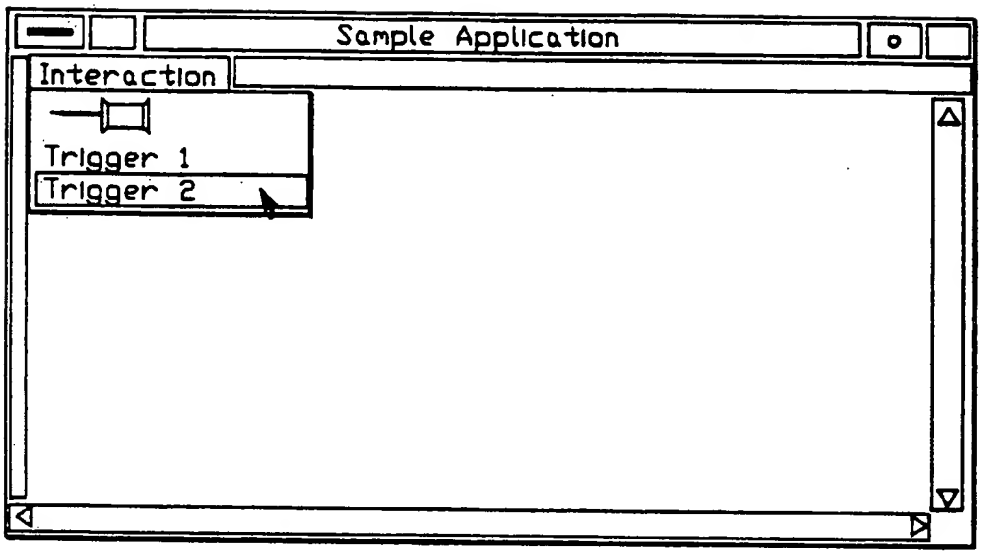


FIG.—10

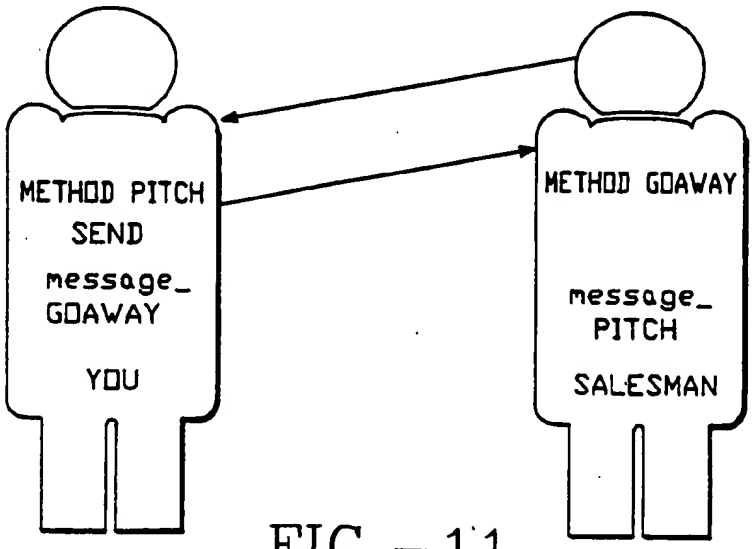


FIG.—11

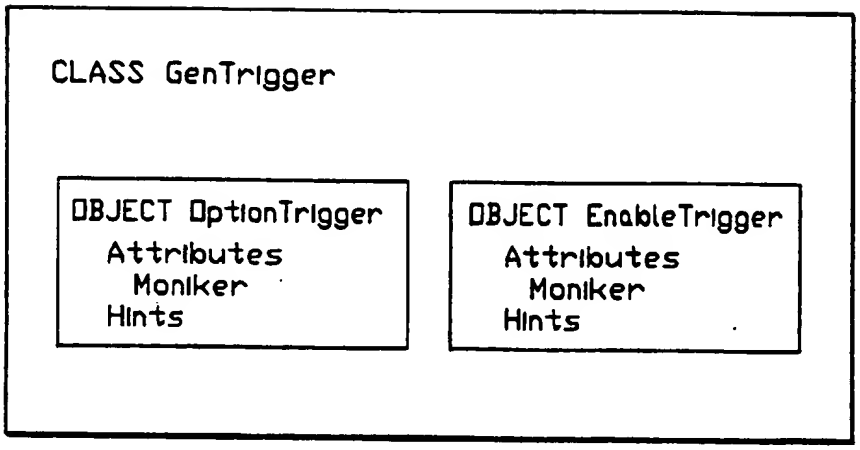

CLASS GenTrigger
OBJECT OptionTrigger
Attributes
Moniker
Hints
OBJECT EnableTrigger
Attributes
Moniker
Hints

FIG.—12

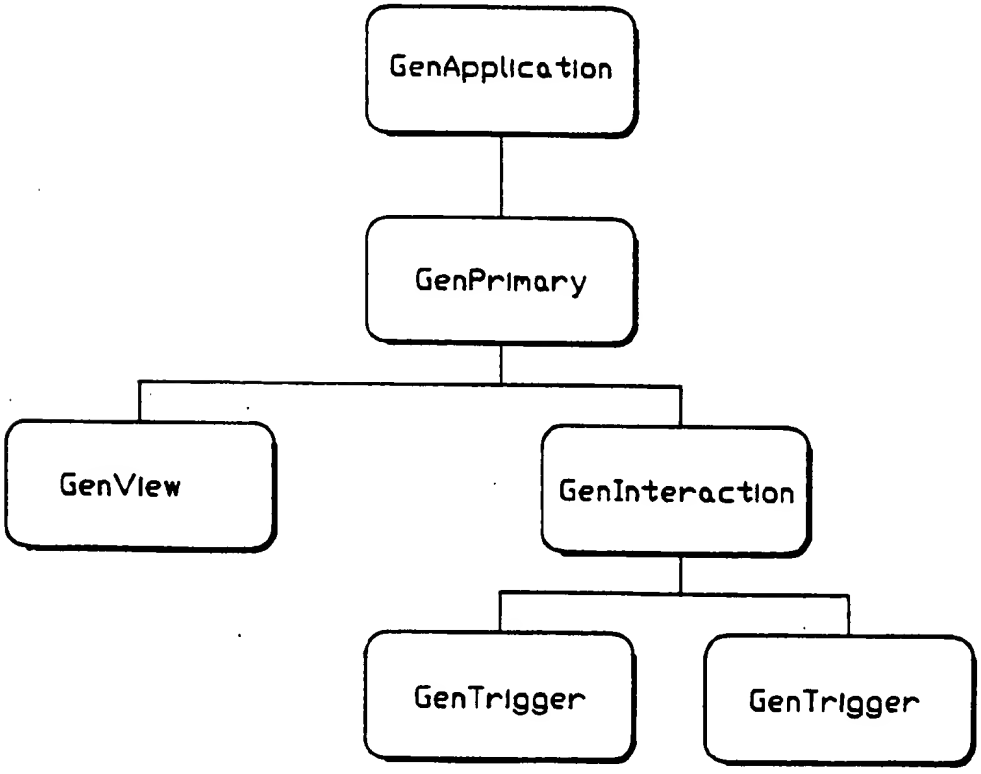

GenApplication
GenPrimary
GenView
GenInteraction
GenTrigger
GenTrigger

FIG.—13

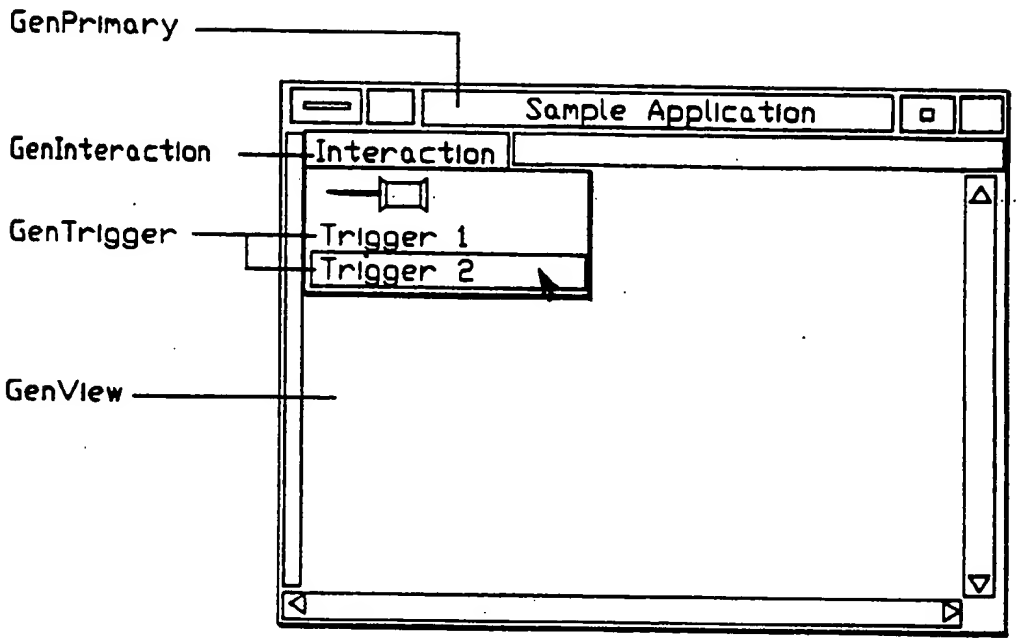

GenPrimary
GenInteraction
GenTrigger
GenView
Sample Application
Interaction
Trigger 1
Trigger 2

FIG.—14

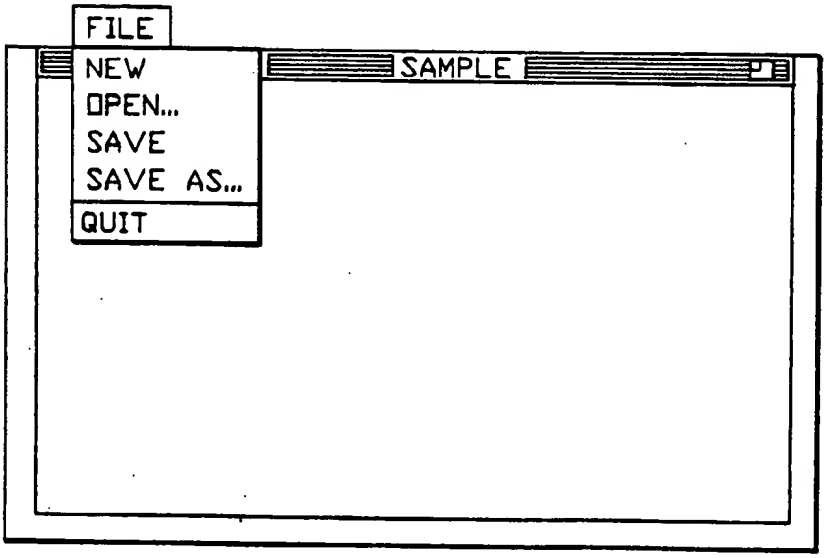

FILE
NEW
OPEN...
SAVE
SAVE AS...
QUIT
SAMPLE

FIG.—15

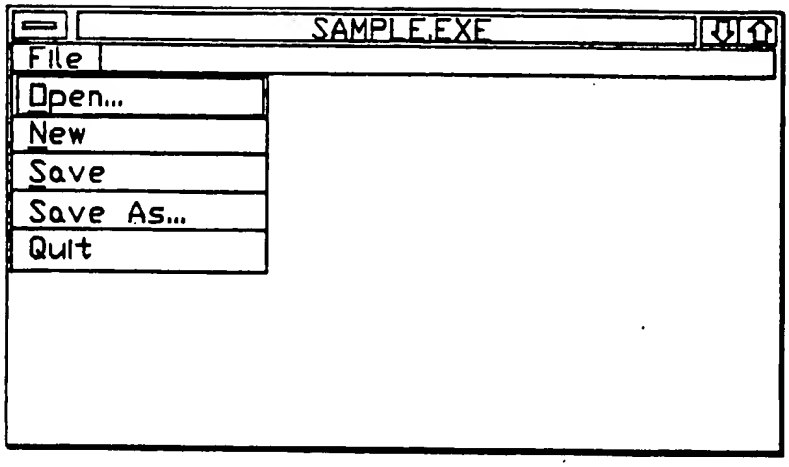


FIG.—16

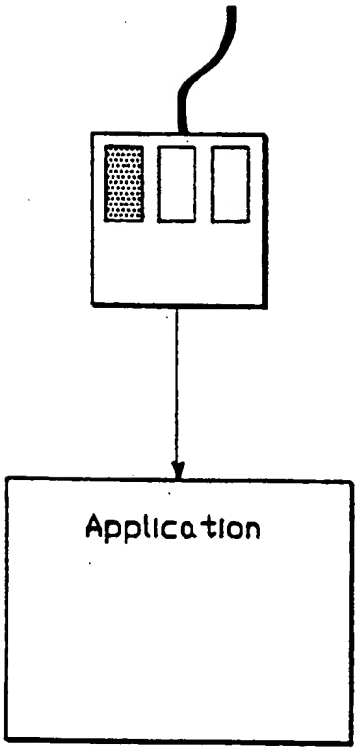


FIG.—17

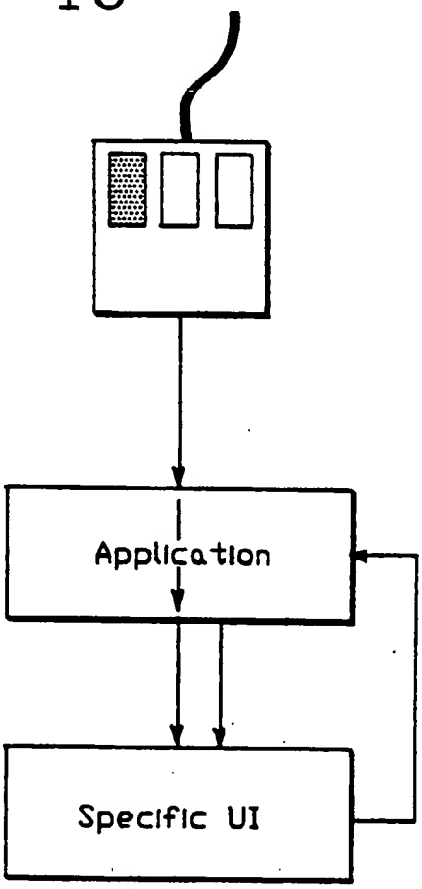


FIG.—18

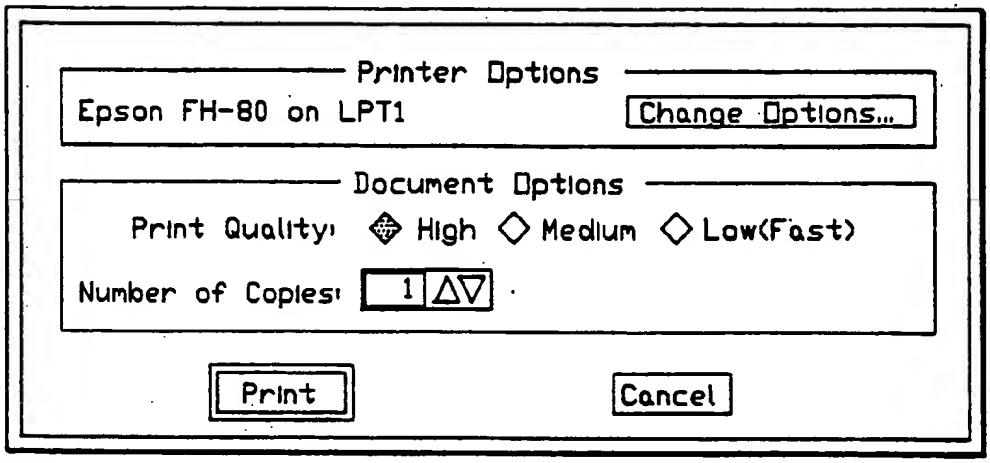

Printer Options
Epson FH-80 on LPT1
Change Options...
Document Options
Print Quality: High Medium Low(Fast)
Number of Copies: 1
Print
Cancel

FIG.—19

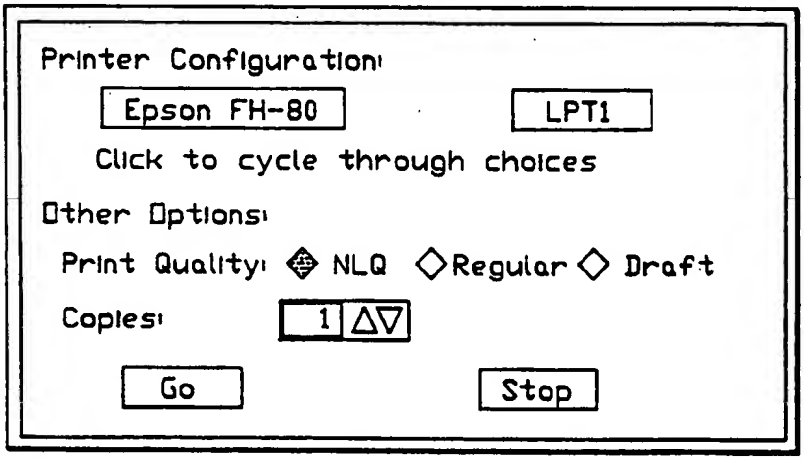

Printer Configuration:
Epson FH-80
LPT1
Click to cycle through choices
Other Options:
Print Quality: NLQ Regular Draft
Copies: 1
Go
Stop

FIG.—20

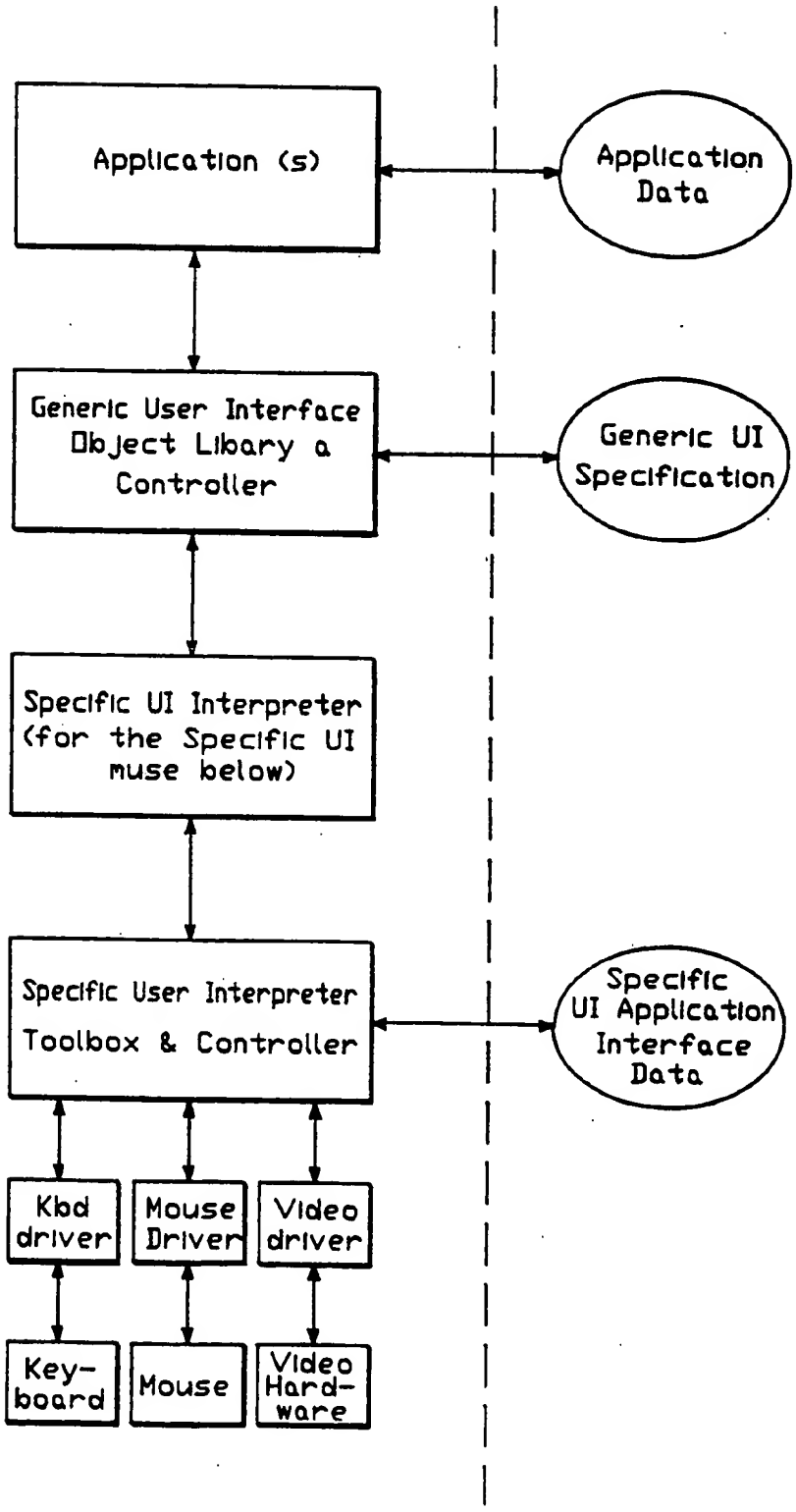

Application (s)
Application Data
Generic User Interface Object Libary a Controller
Generic UI Specification
Specific UI Interpreter (for the Specific UI muse below)
Specific User Interpreter Toolbox & Controller
Specific UI Application Interface Data
Kbd driver
Mouse Driver
Video driver
Key-board
Mouse
Video Hard-ware

FIG.—21

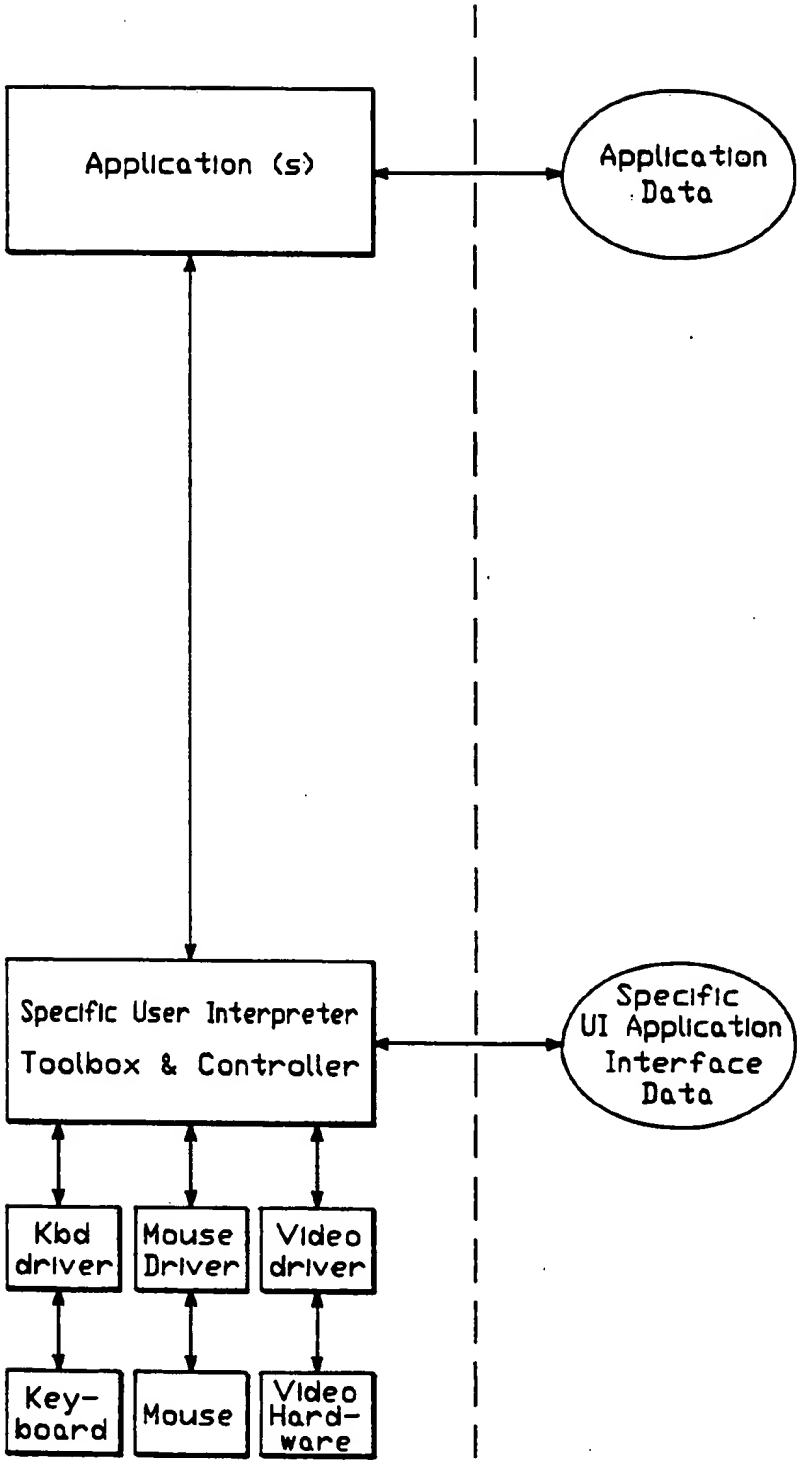


FIG.—22
(Prior Art)

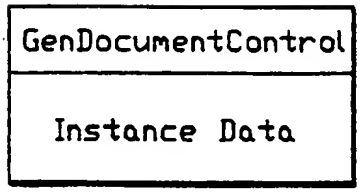


FIG.—23

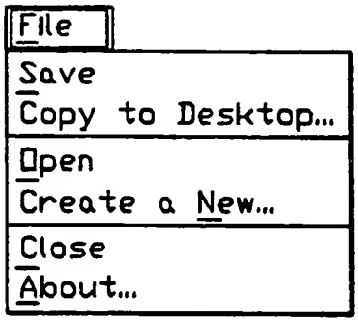


FIG.—24

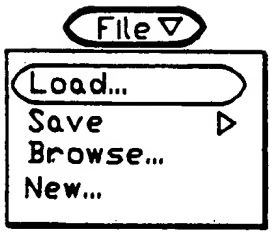


FIG.—25

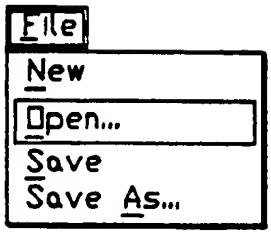


FIG.—26

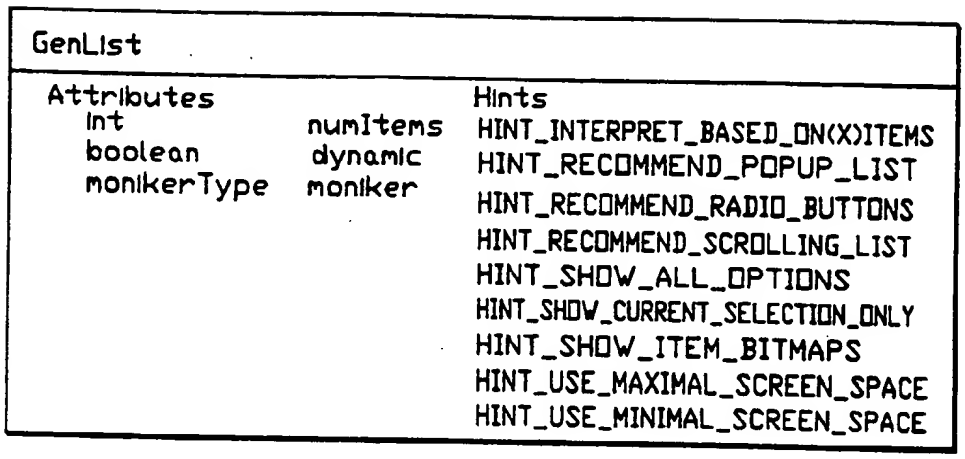


FIG.—27

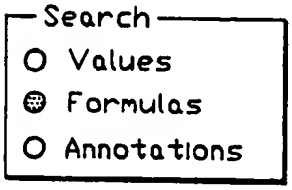


FIG.—28

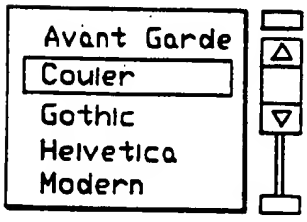


FIG.—29

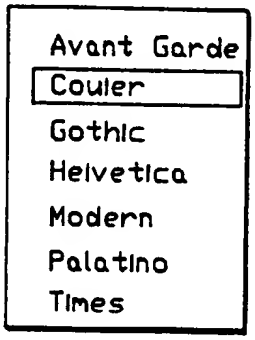


FIG.—30

Possible Gadget Choices	Example	Style Guide Interpretation Rule
Abbreviated Menu Button * occupies small amount of screen space * recommended for ≤ 16 items * shows current selection only * shows all options while choosing	Fruit: Apple Fruit: Apple Banana Kiwi Necterine Orange Pear Tomato	Abbreviated Menu Button may be use IF (6 ≤ items ≤ 16) OR (items ≤ 16) AND (HINT_RECOMMEND_POPUP_LIST OR HINT_USE_MINIMAL_SCREEN_SPACE OR HINT_SHOW_CURRENT_SELECTION_ONLY OR (HINT_INTERPET_BASED_ON(X)ITEMS AND 6 ≤ X ≤ 16)
Exclusive Setting * occupies large amount of screen space * shows all selection options at all times * recommended for 2-5 options, up to 10	Size: Small Medium Large	Exclusive Setting may be use IF (2 ≤ items ≤ 5) OR (items ≤ 10) AND (HINT_RECOMMEND_RADIO_BUTTONS OR HINT_USE_MAXIMAL_SCREEN_SPACE OR HINT_SHOW_ALL_OPTIONS OR (HINT_INTERPET_BASED_ON(X)ITEMS AND 2 ≤ X ≤ 5))
Scrolling List * occupies medium to large amount of screen space * handles any number of items * default of 5 entries showing at a time	Font: Avant Garde Couier Gothic Helvetica Modern	Scrolling List may be use IF (items ≤ 17) OR dynamic number of items OR HINT_RECOMMEND_SCROLLING_LIST OR (HINT_INTERPET_BASED_ON (X) ITEMS AND X ≤ 17)

FIG.-31

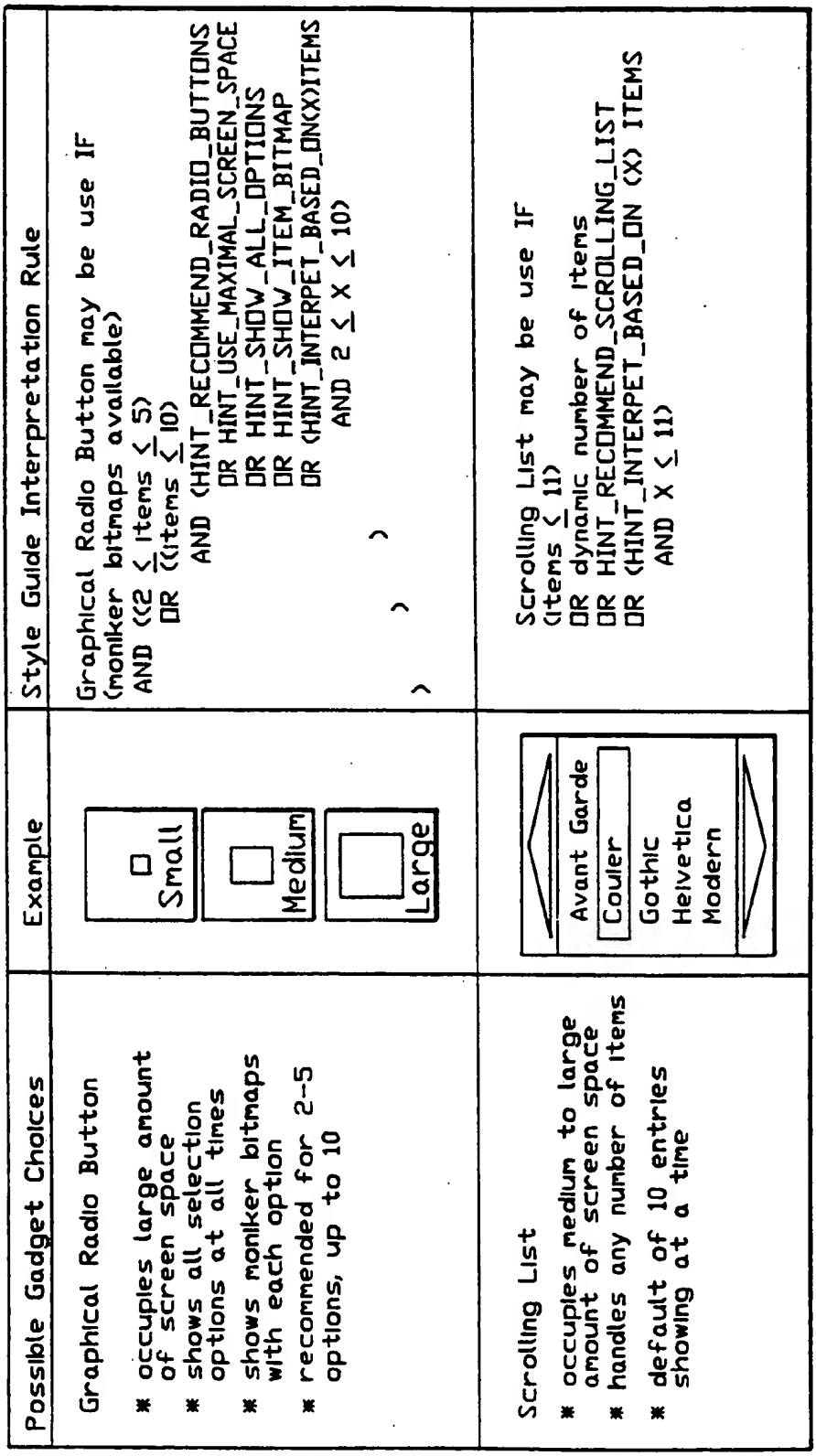

Possible Gadget Choices
Example
Style Guide Interpretation Rule
Graphical Radio Button
* occupies large amount of screen space
* shows all selection options at all times
* shows moniker bitmaps with each option
* recommended for 2-5 options, up to 10
Small
Medium
Large
Graphical Radio Button may be use IF
(moniker bitmaps available)
AND ((2 ≤ items ≤ 5)
OR ((items ≤ 10)
AND (HINT_RECOMMEND_RADIO_BUTTONS
OR HINT_USE_MAXIMAL_SCREEN_SPACE
OR HINT_SHOW_ALL_OPTIONS
OR HINT_SHOW_ITEM_BITMAP
OR (HINT_INTERPET_BASED_ON(X)ITEMS
AND 2 ≤ X ≤ 10)
)
)
)
Scrolling List
* occupies medium to large amount of screen space
* handles any number of items
* default of 10 entries showing at a time
Avant Garde
Couier
Gothic
Helvetica
Modern
Scrolling List may be use IF
(items ≤ 11)
OR dynamic number of items
OR HINT_RECOMMEND_SCROLLING_LIST
OR (HINT_INTERPET_BASED_ON (X) ITEMS
AND X ≤ 11)

FIG.—32

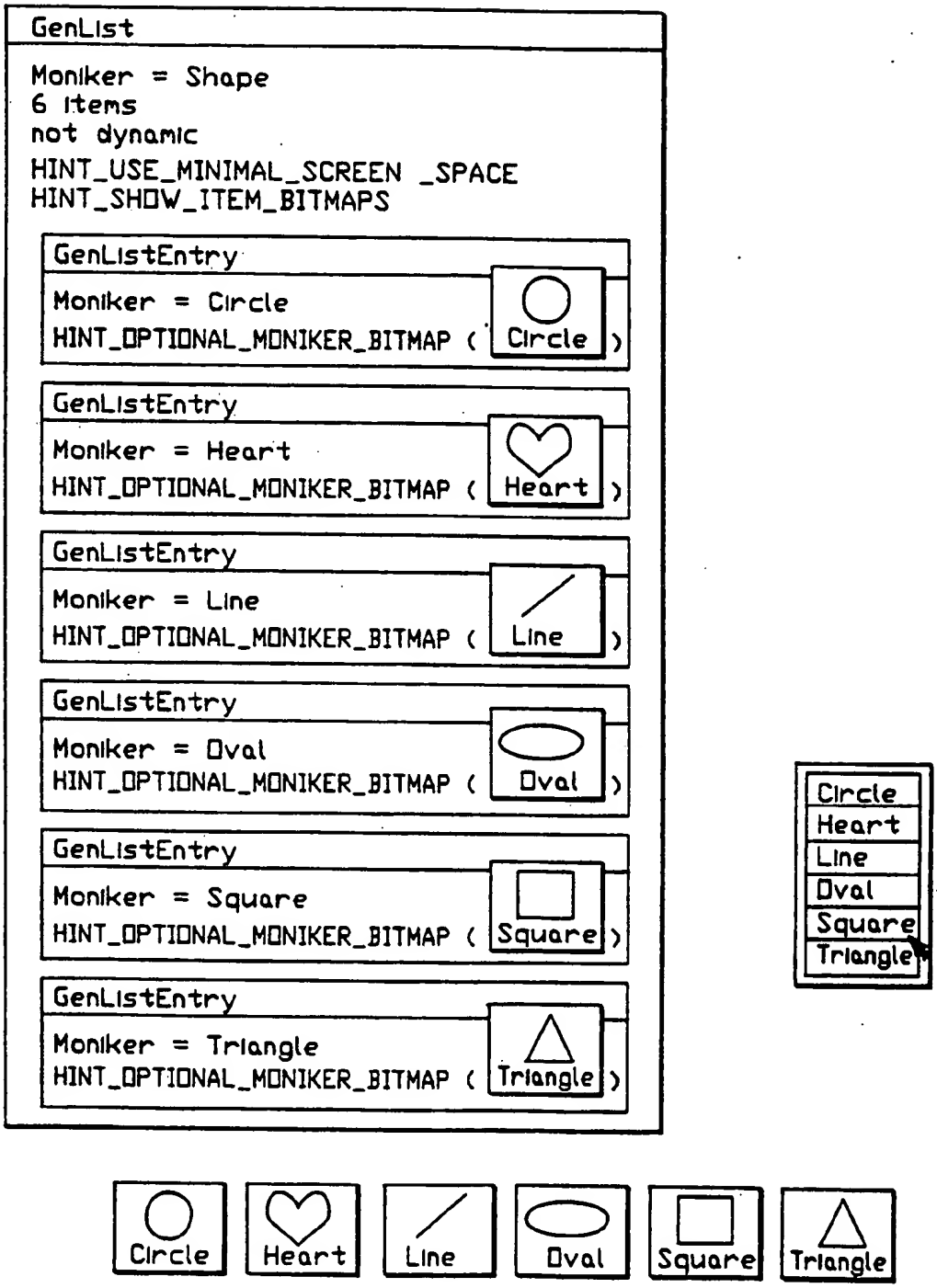

GenList
Moniker = Shape
6 items
not dynamic
HINT_USE_MINIMAL_SCREEN _SPACE
HINT_SHOW_ITEM_BITMAPS
GenListEntry
Moniker = Circle
HINT_OPTIONAL_MONIKER_BITMAP (Circle)
GenListEntry
Moniker = Heart
HINT_OPTIONAL_MONIKER_BITMAP (Heart)
GenListEntry
Moniker = Line
HINT_OPTIONAL_MONIKER_BITMAP (Line)
GenListEntry
Moniker = Oval
HINT_OPTIONAL_MONIKER_BITMAP (Oval)
GenListEntry
Moniker = Square
HINT_OPTIONAL_MONIKER_BITMAP (Square)
GenListEntry
Moniker = Triangle
HINT_OPTIONAL_MONIKER_BITMAP (Triangle)
Circle
Heart
Line
Oval
Square
Triangle
Circle
Heart
Line
Oval
Square
Triangle

FIG.—33

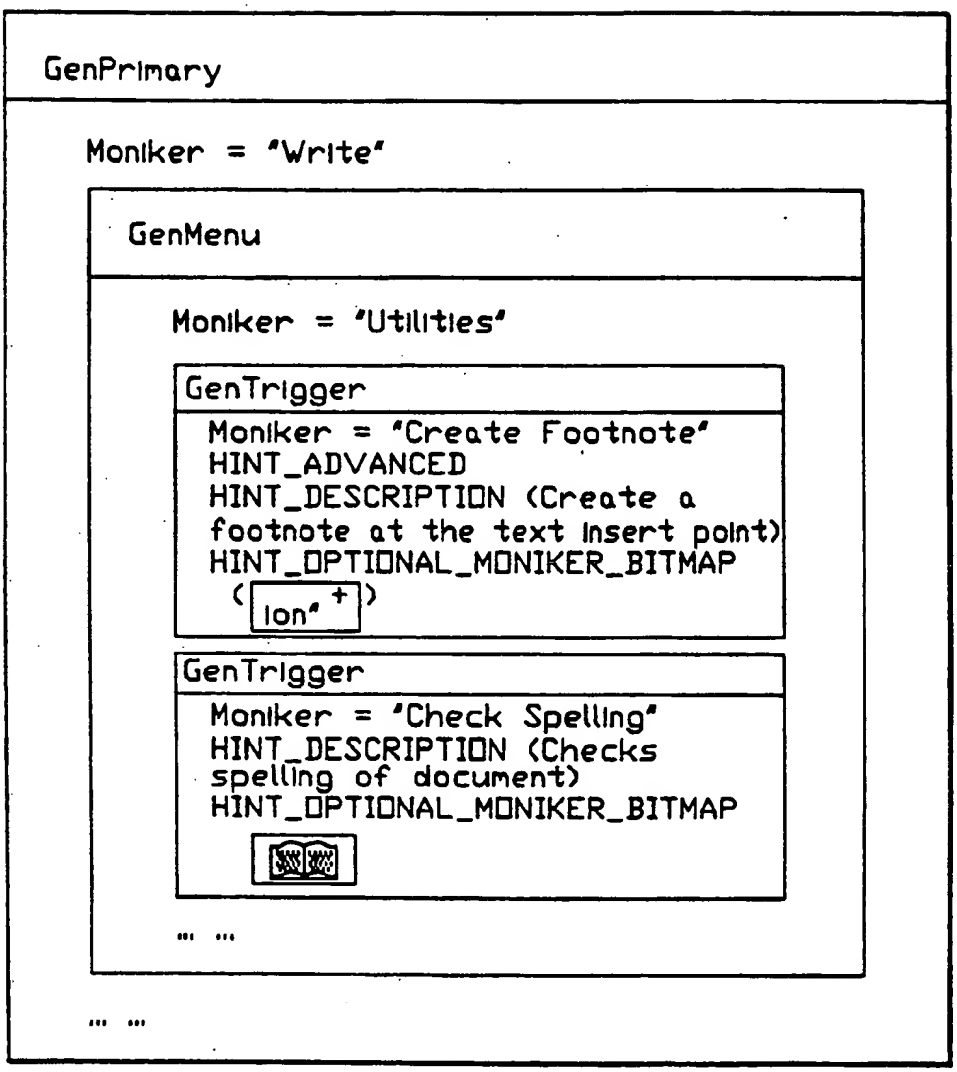

GenPrimary
Moniker = "Write"
GenMenu
Moniker = "Utilities"
GenTrigger
Moniker = "Create Footnote"
HINT_ADVANCED
HINT_DESCRIPTION (Create a footnote at the text insert point)
HINT_OPTIONAL_MONIKER_BITMAP
(Ion" +)
GenTrigger
Moniker = "Check Spelling"
HINT_DESCRIPTION (Checks spelling of document)
HINT_OPTIONAL_MONIKER_BITMAP

FIG.—34

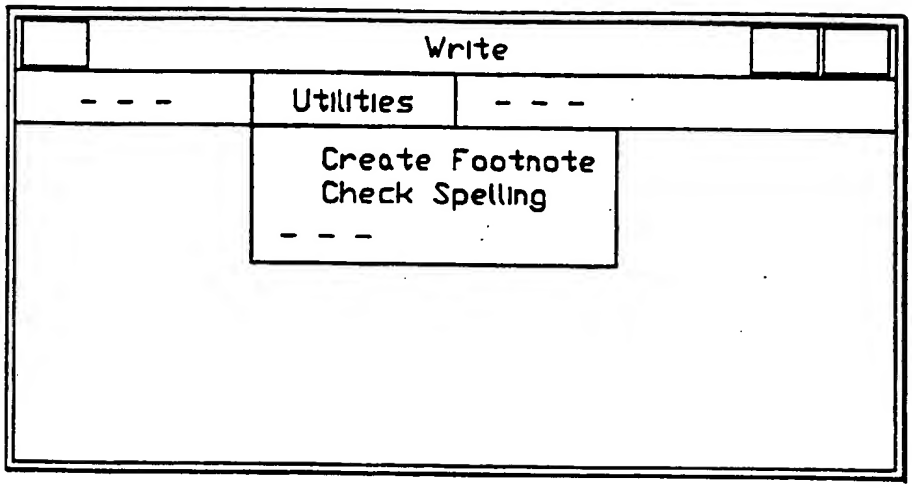

Write
- - -
Utilities
- - -
Create Footnote
Check Spelling
- - -

FIG.—35

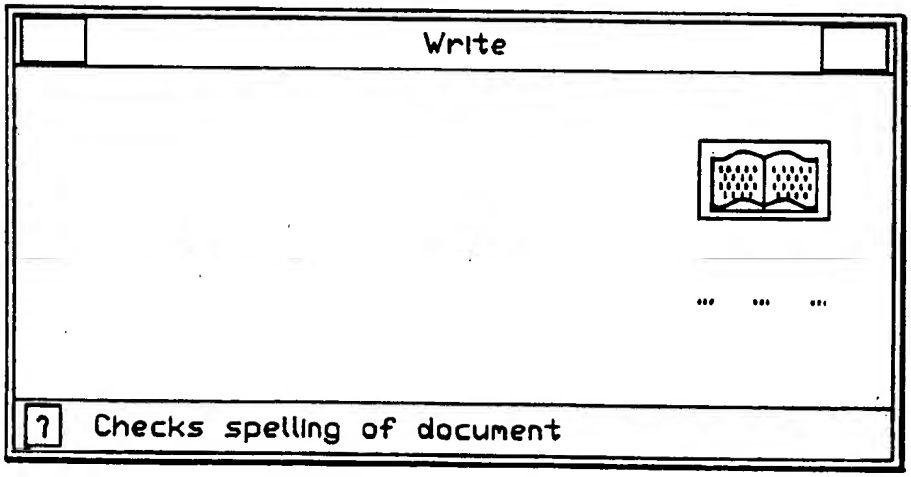

Write
Checks spelling of document

FIG.—36

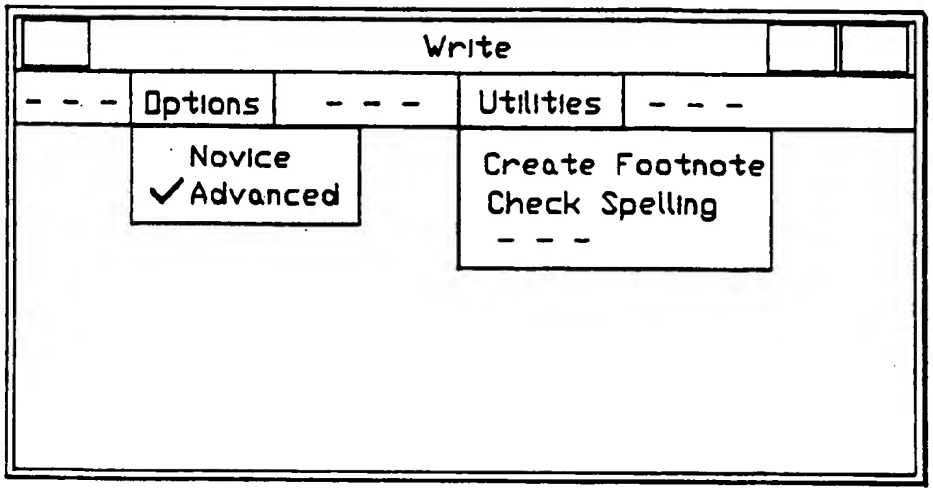

Write
- - -
Options
- - -
Utilities
- - -
Novice
✓Advanced
Create Footnote
Check Spelling
- - -

FIG.—37

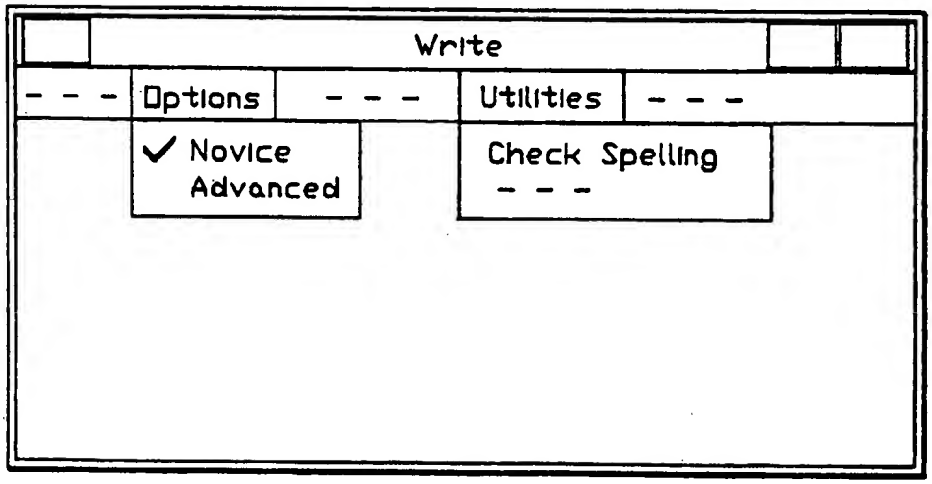

Write
- - -
Options
- - -
Utilities
- - -
✓ Novice
Advanced
Check Spelling
- - -

FIG.—38

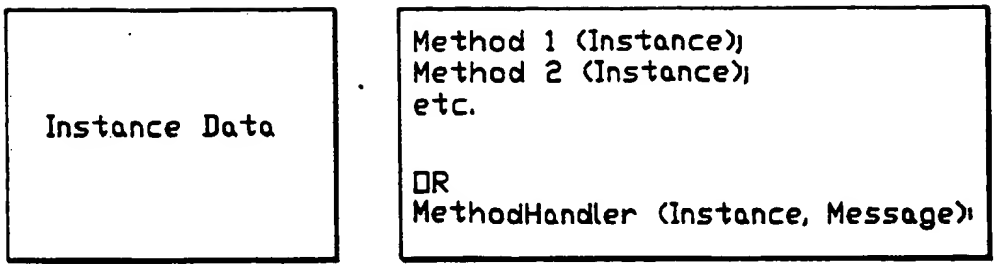

Instance Data
Method 1 (Instance);
Method 2 (Instance);
etc.
OR
MethodHandler (Instance, Message);

FIG.—39

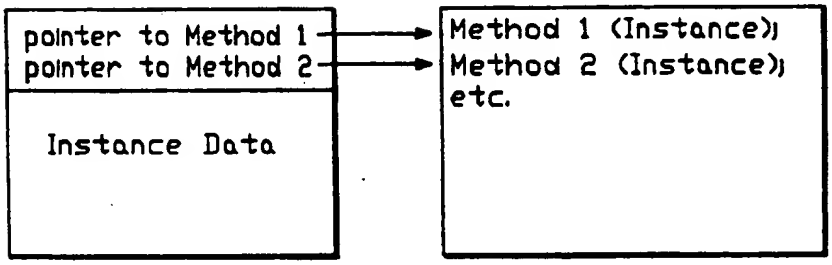

pointer to Method 1
pointer to Method 2
Instance Data
Method 1 (Instance);
Method 2 (Instance);
etc.

FIG.—40

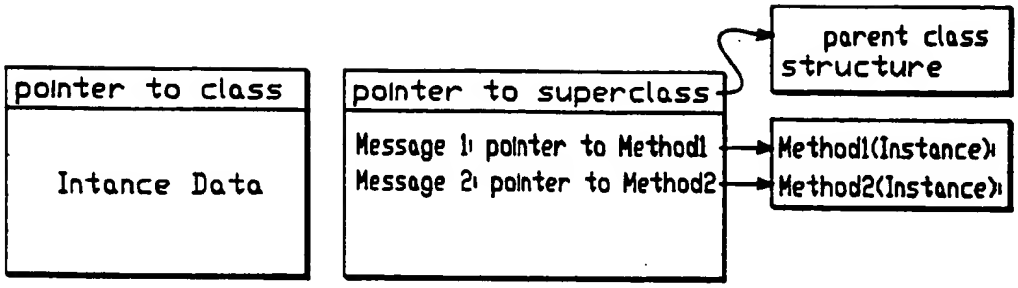


FIG.—41

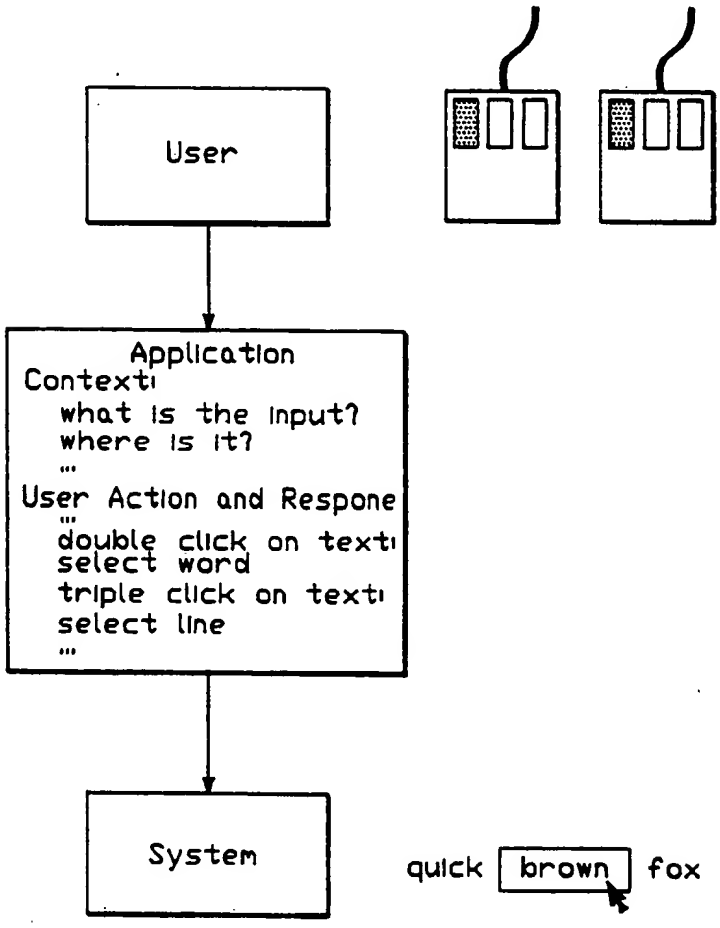


FIG.—42
(Prior Art)

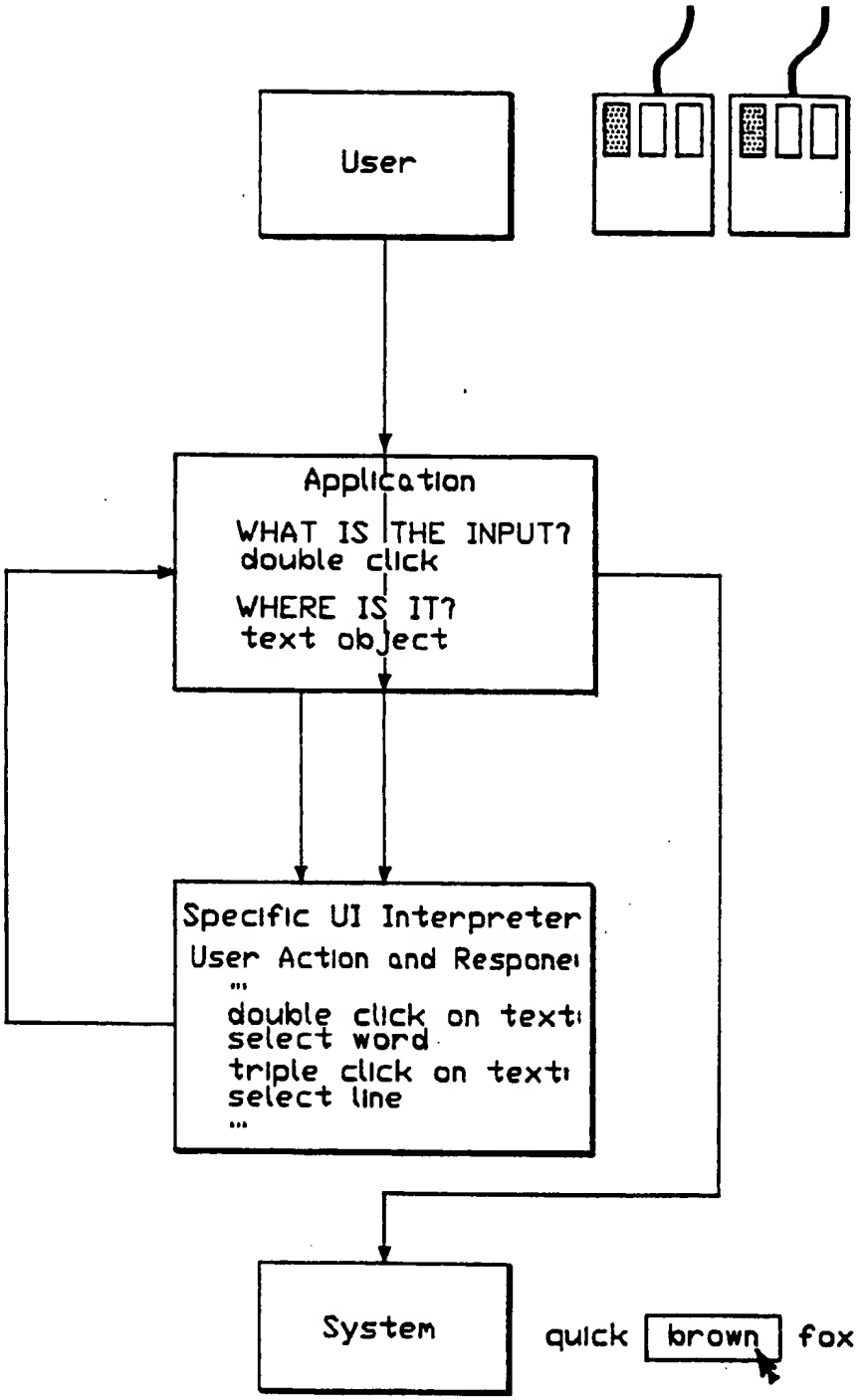

User
Application
WHAT IS THE INPUT?
double click
WHERE IS IT?
text object
Specific UI Interpreter
User Action and Response
...
double click on text:
select word
triple click on text:
select line
...
System
quick
brown
fox

FIG.—43

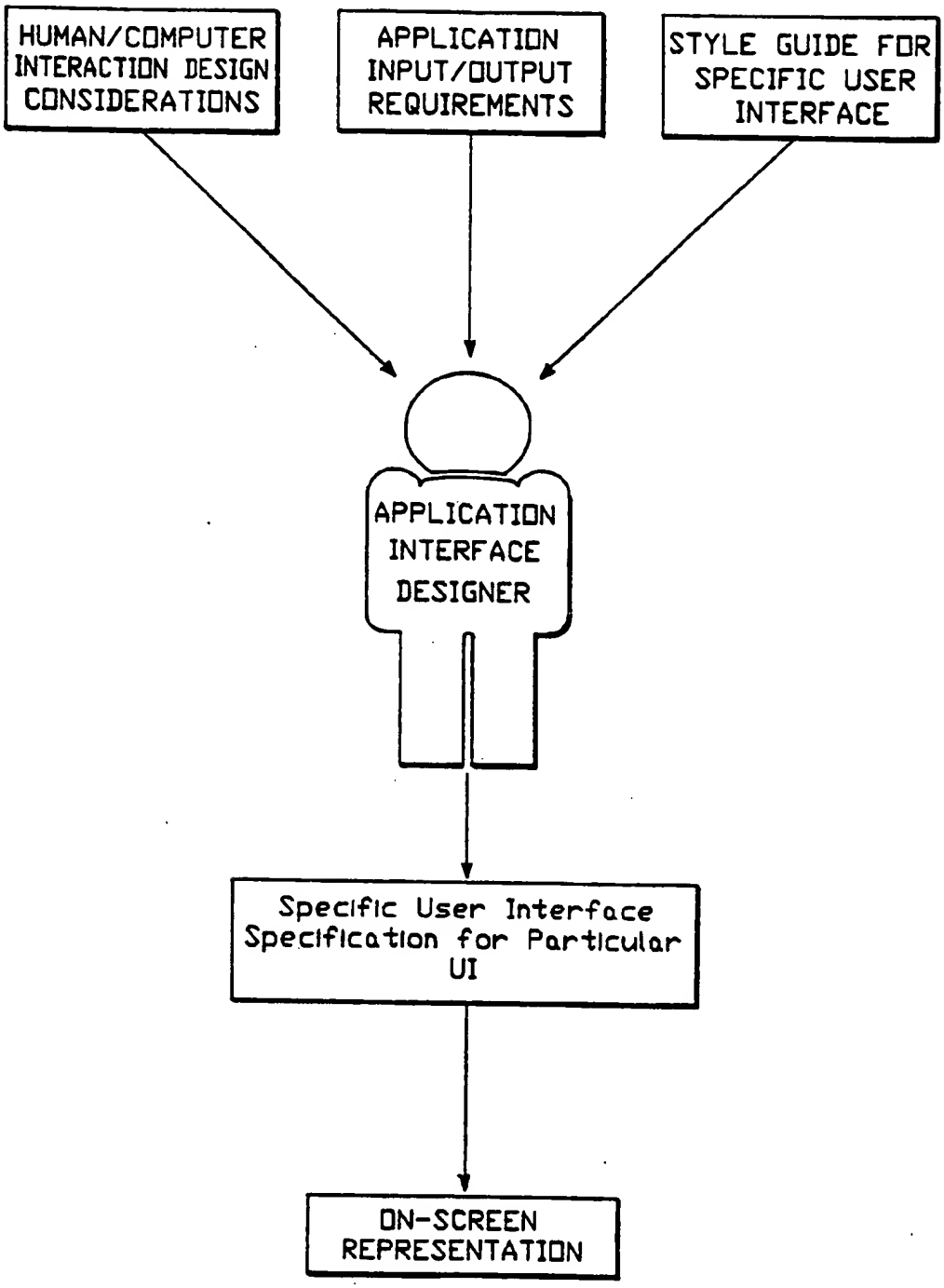


FIG.—44
(Prior Art)

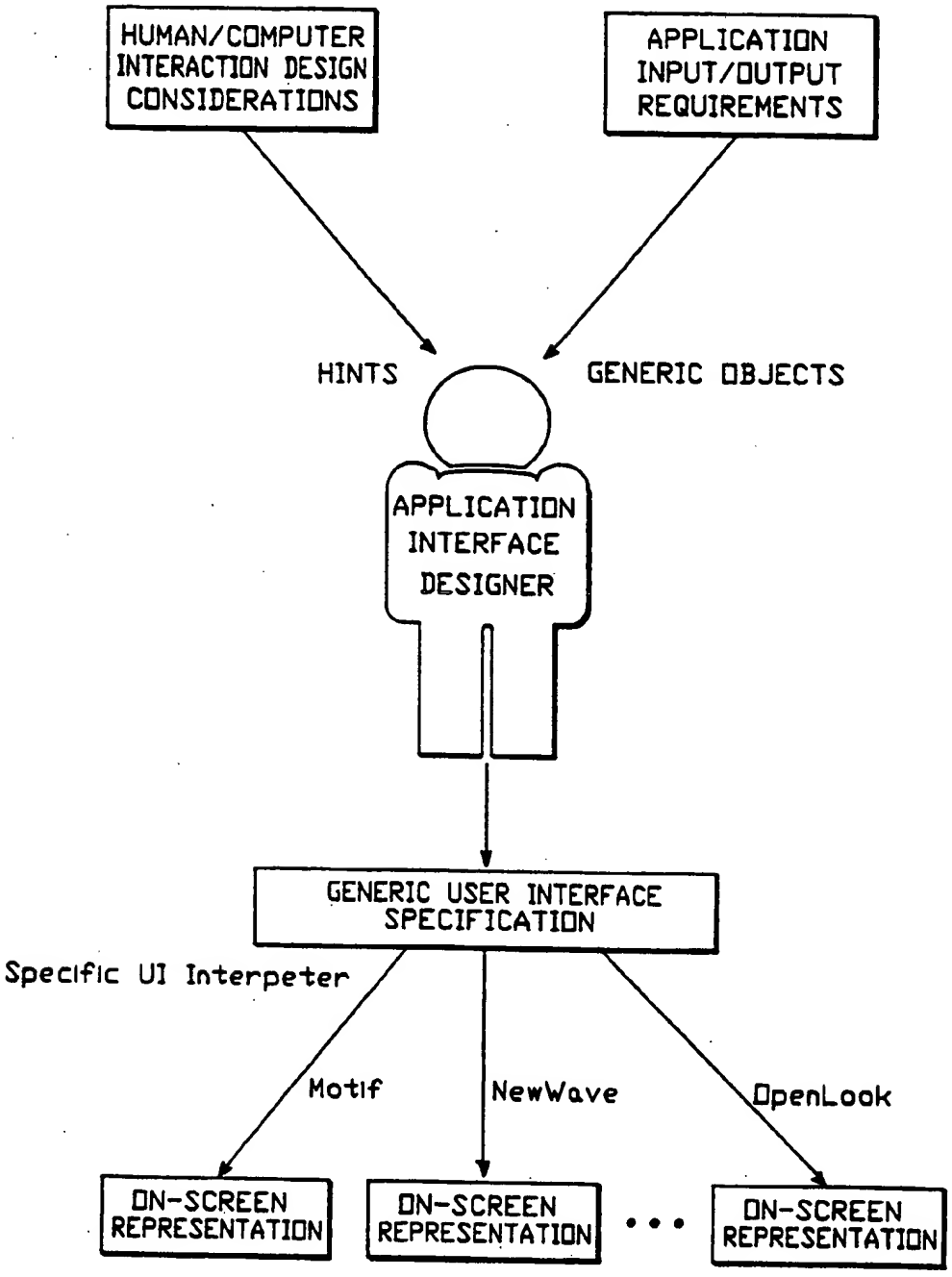

HUMAN/COMPUTER INTERACTION DESIGN CONSIDERATIONS
APPLICATION INPUT/OUTPUT REQUIREMENTS
HINTS
GENERIC OBJECTS
APPLICATION INTERFACE DESIGNER
GENERIC USER INTERFACE SPECIFICATION
Specific UI Interpeter
Motif
NewWave
OpenLook
ON-SCREEN REPRESENTATION
ON-SCREEN REPRESENTATION
ON-SCREEN REPRESENTATION

FIG.—45

FIG.—46

FIG.—47

PROCESS OF DESIGNING USER'S INTERFACES FOR APPLICATION PROGRAMS

This application is a continuation of application Ser. No. 07/681,079, filed Apr. 5, 1991, now abandoned, which is a continuation-in-part of application Ser. No. 07/586,861, filed Sep. 24, 1990, now abandoned.

BACKGROUND OF THE INVENTION

1. Field of the Invention

In general, the present invention relates to an application program operating in a computer and, more particularly, the invention relates to a process for designating a user interface of an application program.

2. Description of the Related Art Applications

An application (or program) is a tool which allows a person to use a computer to accomplish a task. For example, a word processor provides the user of a computer with a way to write, store, and print out letters. A drawing program allows him to create charts, diagrams, and organizational charts. As far as the user is concerned, the application is the interface between him and the computer hardware. However, from the application's perspective, there is another layer.

Operating Systems

An operating system is a program which acts as an interface between applications and the computer hardware. It provides an environment in which a user may execute programs. Operating systems attempt to make the computer system easy and efficient to use. Operating systems in conjunction with the computer hardware are often called environments. These principles are discussed by James Peterson and Abraham Silberschatz in "Operating System Concepts".

User Interfaces

A user interface (UI) is a set of rules and conventions by which a computer system communicates with the person operating it. Initially, operating systems (such as UNIX or MS-DOS) featured text-based command line interfaces. Users were expected to use and remember complicated, forgettable commands such as "enscript-2Gr-Plw." Different applications all had different user interfaces—to print the current document, a user might have to press the function key F7 in a word processor and the keys Ctrl-Alt-P in a database program. Computers were difficult to learn, difficult to use, and, worst of all, inconsistent. In the pursuit of the often-coined property known as "user friendliness," much work was done in terms of improving user interfaces. Just as the personal computer market as a whole is changing rapidly and drastically, so too are user interface standards. Through the years, operating systems have evolved from complicated text-based, command line interfaces, like UNIX or MS-DOS, to graphical, windowing interfaces such as the Apple Macintosh and Microsoft Windows. These new graphical user interfaces (GUIs) feature menus, buttons, and windows accessed by a mouse. The graphical, intuitive nature of these interfaces solved many of the problems inherent in earlier operating systems. GUIs typically provide a large tool kit of user interface gadgets such as windows, buttons, and menus. Applications make use of these UI items to implement their interaction with the user. In order to avoid inconsistent application interfaces, companies develop rules and conventions for using the UI gadgets. Documents known as style guides are provided in an attempt to instruct application designers in the appropriate usage of the user interface gadgets offered by a system (see "Style Guides" for more detailed information). Some examples of such user interface standards are OSF/Motif, OpenLook, CUA, NewWave, and Macintosh. Each of these standards shall be referred to herein as a specific user interface.

However, even applications developed for a "user friendly" environment like Windows or Macintosh sometimes can be difficult to use. As applications have become more and more powerful, some have also become more and more difficult to use. There are so many fascinating and complex things users can do with these new programs that it can be very difficult to create a user interface that is always easy to use. A new concept in the GUI community attempts to come to terms with this problem. It is the scalable graphic user interface. Such a GUI allows the same applications to be accessed at various levels of functionality. These levels range from an appliance mode, where users are only required to push a few buttons, to a novice computer interface (such as Tandy's Deskmate TM), to a full-fledged professional graphic user interface like the Motif TM interface. Users, as their skills and needs grow, may simply switch interface levels to access more powerful features. So, for example, if users only want to quickly type a letter or envelope, they do not have to wade through a program designed to produce newsletters involving multiple columns of text running from page to page and graphics placed randomly throughout the document. They can merely run the word processor in appliance mode and type a simple letter without having to set many different options and to pick their way through a number of extra features (See "Style Guides" for how scalability relates to style guides).

Applications Development

Applications have always been difficult and time-consuming to develop. However, because of the volatile and diverse nature of the computer software industry, creating applications which execute under different specific user interface standards often can be exceptionally challenging. In the past, much or all of the application often was rewritten in order to conform to the various standards, and each version often was offered for sale separately.

Some applications have implemented the scalable GUI concept to some extent. Programs such as Microsoft Word have "full and short menu" modes; novices may choose "short menus," which hides advanced functionality by simply removing advanced commands from the main menu. The user still has to contend with multiple windows and pull-down menus, difficult enough concepts in their own right. However, very few programs even feature this limited scalability. Usually, if users would like both a simple word processor and an advanced word processor, they would have to purchase two separate packages (in fact, some software publishers offer several similar packages of varying complexity in their product line.

Conventional Application Design Process

The typical process of writing an application is as follows. An environment is chosen (e.g., UNIX workstation running Motif). The functional goals of the program are specified (e.g., a powerful word processor). Then the user accessibility goals are specified (e.g.,

must be easy and intuitive to use and follow Motif guidelines). A programmer or team of programmers implements the functionality, and a human interface expert or team (perhaps the same programmers) designs the user interface to conform with the style guide for the environment.

We will focus on the user interface designer. The interface designer balances human/computer interaction design considerations, application input and output requirements, and the style guide for the specific user interface to create a specific interface description (typically in the form of user interface objects with attributes) for the application. The graphical user interface system software then implements this specific description, creating an on-screen representation.

If an operating system could somehow interpret the user interface needs of an application and provide a good implementation of one or more style guides, both application developers and users would benefit.

SUMMARY OF THE INVENTION

It is an objective of the present invention to provide a new process for generating on-screen application interface for an application program.

It is another objective of the present invention to provide a new process for generating user interfaces for application programs.

It is another objective of the present invention to provide a new process for generating the on-screen application interface in such a way that the application is largely independent of changes in specific user interfaces. Developing an application is a challenging and time-consuming project. One essential aspect of this process is the design and implementation of the application's user interface. In the traditional process of interface design, the developer defines application input-/output requirements and a list of human-to-computer design considerations associated with those requirements. Referring to the style guide for the specific user interface (e.g. Motif, OpenLook, or Windows) for which the application is being designed, the designer melds all three considerations together when defining the user interface. He makes selections from the gadget toolkit offered by the specific UI and decides how those objects should be arranged on the screen. These selections and decisions are made with an eye toward subjective design considerations. The exact final interface design is then stored in data structures, which are later faithfully rendered on screen by system software. To run the application under a different specific UI, the design process would have to be repeated from scratch, yielding a new executable version of the application.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 illustrates a conventional application design process.

FIG. 2 illustrates a application design process according to the present invention.

FIG. 3 illustrates a dialogue box.

FIG. 4 illustrates a menu.

FIG. 5 illustrates pixels on a screen.

FIG. 6 illustrates a scroll bar.

FIG. 7 illustrates a scrolling list.

FIG. 8 illustrates a submenu.

FIG. 9 illustrates hierarchy of objects in tree data structure.

FIG. 10 illustrates a window.

FIG. 11 illustrates the salesman example.

FIG. 12 illustrates a GenTrigger class of generic user interface object and two objects in that class.

FIG. 13 illustrates a sample generic user interface tree.

FIG. 14 illustrates an on-screen realization of the sample user interface tree of FIG. 13.

FIG. 15 illustrates a sample user interface screen of a prior art Macintosh application.

FIG. 16 illustrates a sample user interface screen of a prior art OS/2 application.

FIG. 17 illustrates a conventional user interaction.

FIG. 18 illustrates a user interaction according to the present invention.

FIG. 19 illustrates an easy to use layout for print dialogue box that can be more difficult to use.

FIG. 20 illustrates a poor layout for print dialogue box.

FIG. 21 illustrates a computer system incorporating elements in accordance with the present invention.

FIG. 22 illustrates a prior art computer system.

FIG. 23 illustrates a document control object.

FIG. 24 illustrates a NewWave TM interpretation of the document control object of FIG. 23.

FIG. 25 illustrates an OpenLook TM interpretation of the document control object of FIG. 23.

FIG. 26 illustrates a Motif TM interpretation of the document control object of FIG. 23.

FIG. 27 illustrates a list object and some possible hints that can be used as instance data for that object.

FIG. 28 illustrates a NewWave TM interpretation of the list object of FIG. 27.

FIG. 29 illustrates an OpenLook TM interpretation of the list object of FIG. 27.

FIG. 30 illustrates a Motif TM interpretation of the list object of FIG. 27.

FIG. 31 illustrates a style guide interpreter that provides three possible gadget choices (Abbreviated Menu Button, Exclusive Settings and Scrolling List) with an example of a screen display and style guide interpreter interpretation rules for each.

FIG. 32 illustrates a style guide interpreter that provides two possible gadget choices for a hypothetical user interface (Graphical Radio Buttons and Scrolling List) with an example of a screen display and style guide interpreter interpretation rules for each.

FIG. 33 illustrates a generic user interface specification for the GenList object and an OpenLook TM user interface interpretation and a hypothetical user interface interpretation of the generic GenList object.

FIG. 34 illustrates a sample generic user interface specification.

FIG. 35 illustrates an interpretation of an object having the specification of FIG. 34 under Motif TM or OpenLook TM.

FIG. 36 illustrates an interpretation of an object having the specification of FIG. 34 under a hypothetical user interface style guide.

FIG. 37 illustrates an interpretation of an object having the specification of FIG. 34 under a hypothetical user interface of the future in an advanced mode.

FIG. 38 illustrates an interpretation of an object having the specification of FIG. 34 under a hypothetical user interface of a future in a novice mode.

FIG. 39 illustrates an implementation of the principles of the invention using procedural programming rules.

FIG. 40 illustrates an implementation of the principles of the invention using objects using pointers to methods.

FIG. 41 illustrates an implementation of the principles of the invention using objects having class pointers to class structures.

FIG. 42 illustrates a prior art user interaction with an application to produce an action in a computer system.

FIG. 43 illustrates a user interaction with an application in accordance with the present invention to produce an action in a computer system.

FIG. 44 illustrates prior art development and use of a specific user interface specification for a particular user interface.

FIG. 45 illustrates development and use of a generic user interface specification for use with any of multiple specific user interface interpreters (Motif TM, NewWave TM and OpenLook TM, for example), and the use of such generic user interface to produce different on-screen displays using such different specific user interface interpreters.

FIG. 46 illustrates a one-choice element of a control area or a menu, used in various ways such as to execute commands, display pop up windows, and display means.

FIG. 47 illustrates a non-exclusive setting that shows a checkmark in a square box when a setting is chosen.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENT

Before explaining the principles in the present invention, it is useful to define a group of terminologies as follows.

gadget toolkit

set of components such as windows, menus, buttons, scrolling lists, radio buttons, scroll bars, etc. A portion of every specific user interface's style guide concerns itself with the enumeration, definition, and uses of these components

generic UI object class

group of generic UI objects with identical types of data and methods

generic UI object

uI component representing an input/output need of an application (as opposed to a visual specification such as a scrolling list). Examples include document control, exclusive list selection, and viewing areas.

generic UI object library

set of generic UI objects and hints available for specifying an interface independent of any particular gadget toolkit

generic user interface specification

interface designed for a particular application based on objects from the generic UI object library, including the selection and organization of objects and hints

generic to specific UI interpreter (UI Interpreter)

software which interprets a generic UI specification to create the on-screen representation of the application in such a way that a specific UI's style guide requirements and recommendations are met

hint

embodiment of human/computer interface criteria for an application, stored digitally. Examples of such criteria follow:

"infrequently used feature"

"advanced feature"

"should be displayed as large as possible"

specific user interface

the look and feel ONLY of a particular user interface specification, such as Motif, Open Look, Windows, or Macintosh, as denoted by that user interface's style guide (i e the end user's perception of the user interface, separated from the API and software)

specific user interface specification

interface designed for a particular application based on gadgets from a specific user interface's gadget toolkit

Conventional Application Design Process

The process of the present invention has redefined how application user interfaces are developed. The illustrative drawings of FIG. 1 shows a representative conventional process for developing an application. In contrast to the conventional process, a designer using a new process in accordance with the present invention does not attempt to define the final, gadget-level interface to the application. Instead, referring to FIG. 2, he selects objects from a generic UI object library based solely on the input/output requirements of the application, and groups them according to function within the application. Subjective design considerations associated with those requirements, which would ordinarily be weighed by the designer in order to pick specific gadgets, are instead stored digitally along with the generic UI objects. The designer's job is done, as everything short of the style guide has been considered and stored as part of the UI specification for the application. This data is later interpreted in software by any one of a number of UI interpreters, which map the selected generic UI objects and hints into an interface implementation which meets the specific UI's style guide requirements. The final interface for the application is then presented on-screen.

GEOS Application Design Process

The process of the present invention allows the same application executable to come up with the look and feel of any number of specific user interfaces, meeting the style guide requirements and recommendations for each. The illustrative drawing of FIG. 2 shows an application development process in accordance with the present invention. The more information about the application's interface requirements and subjective considerations that can be stored in generic object and hint format, the better the interface that can be created for the application when running under UI interpreters for different or new specific user interfaces. Since the generic model essentially decouples the application from its user interface, the application is completely independent of changes in specific user interfaces. The application's user interface is specified solely in terms of common semantic properties rather than specifics of particular UI gadgets, so the application's user interface can be properly constructed and presented under new and different specific user interfaces. New UI interpreters for new style guides can he written after the creation of

an application executable, and the application's user interface will be presented in accordance with the new style guide. What this means is that new, improved user interfaces could add novel and wonderful capabilities far beyond that imagined by the original application designer, simply because functional as well as subjective information about the application's UI needs are stored with the application. Similarly, specific user interfaces intended for users with varying levels of proficiency may be defined, so the very same application executable can also be presented appropriately to both novice, average, and advanced users.

The question then is how to accurately represent, in data, application input/output requirements and subjective design considerations. The GEOS process replaces traditional gadget toolkits with a generic UI object library and stores subjective, descriptive considerations digitally in hints.

Generic UI Object Library vs. Gadget Toolkits

As mentioned before, conventional operating systems provide the developer with gadget toolkits. These toolkits generally attempt to provide a large number of simple as well as sophisticated user interface components. The idea is that given a plethora of low-level building blocks, the interface designer may use, combine, and organize them in such a way that he can balance size, speed, and complexity issues. Unfortunately, this method of defining user describing and storing the core, fundamental input/output needs or capabilities of an application. And it does not necessarily give any indication of the raw subjective information the designer considered in order to select from the gadgets available and to lay them out.

The generic UI object library overcomes these limitations. Input/output needs are abstracted to the highest level possible. Functional needs are identified and placed into distinct categories, called generic UI object classes. The subjective, descriptive thoughts and considerations previously existing solely in the mind of the UI designer are stored as characteristics, known as hints, of the application and its user interface.

The following sections elaborate on how the new model improves substantially upon prior art, on a practical as well as a conceptual level.

Abstractions In Scope

To properly present a given application in any number of different specific user interfaces, it is necessary to abstract many higher-level functional requirements of the application. Otherwise there is a risk that an interface specified for one style guide which might contradict the requirements of another. For instance, two style guides may conflict in their requirements for what must appear in the "File" menu:

Hypothetical Specific UI "A" Example

Style guide "A" requires "File" menu to have these menu items:

New—creates a new document
Open . . . —opens a previously created document
Close—closes an open document (user chooses whether to save changes)
Save—saves changes but does not close document
Save As . . . —save changes under a different name, original document is untouched
Copy To . . . —copies modified document to another file name
Exit—exit from program (user chooses whether to save and close document

Specific UI "B"

Style guide requires "File" menu to have these menu items:

Create—create a new document
Open . . . —open a previously created document
Close—close and save a document
Quit—end program (automatically save document)

A specific UI specification which provides a "File" menu with the items required for specific UI "B" would be an illegal interface for specific UI "A" since menu items are named differently and function differently. This problem is solved by abstracting the fundamental need for "document control" within an application. Most applications have a need to manage and manipulate documents, so the generic UI object library provides a single GenDocumentControlClass object. This object, if chosen for use in an application's interface, stores the abstract concept that the application performs operations on files and therefore needs the user interface to allow the user to manipulate files. The generic UI to specific UI interpreter software for each of the above specific UIs processes the existence of a GenDocumentControlClass object by creating the appropriate file menu, as specified by the style guide.

By contrast, conventional methods of application design often require the developer to create two distinct executables to conform to A and B's style guides. One would contain code to generate the proper File menu with the one type of behavior. The other would contain different code to generate the shorter File menu with another different type of behavior.

Abstraction of Function

A common interface requirement of an application is to let the user choose between a number of different options. Some of the gadgets available in different specific UIs which may be used to accomplish this are:

a scrolling list of items, of which one is highlighted
radio buttons, of which one may be selected (pushed in, like the buttons on a stereo receiver to choose between Tuner, Tape, CD, etc.)
a menu of items, in which the last one selected is checked
a pop-up list, whereby the current selection is shown. Clicking on it brings up a window which shows the rest of the possible selections. Dragging the mouse over the desired item and releasing selects it.

Most specific UIs offer one or more of these options, yet the basic abstract input/output requirement is the same—the user may choose one item out of a list of several. The generic UI object library classifies each of the gadgets above as being functionally identical, and thus provides only the GenListClass object. The choice of specific implementation is left up to the UI interpreter. As a result, applications are not bound to only using a particular gadget. For example, a traditional application might choose to use a scrolling list. A pop-up list might be more appropriate, but suppose it hadn't been invented yet. The application is stuck with the scrolling list because it is hard coded into the program. With the new approach, the application would take advantage of the newest technologies available. Conversely, the GEOS application also works properly under a specific user interface which forbids the use of

scrolling lists, whereas the traditional application would not.

Storage of Subjective Design Considerations

In order for existing and future UI interpreters to make intelligent decisions on the choice of UI gadgetry, the proper information must be available. Beyond the functional behavior that the application needs or expects, the designer knows other, less tangible information. For instance, when implementing a list where the user chooses from a number of different options, the designer might know which of the following general characteristics might apply:

This feature is obscure
This feature is commonly used
This feature is easily understood
This feature is advanced

He would probably know more specific characteristics:

Want to see as many of these items at a time as possible when choosing
Going through each item one at a time while choosing is fine
Want to see all options (selected and unselected) at all times

These subjective pieces of information can be enumerated and appropriate statements "stored" along with generic UI objects. In this case, the GenListClass object would be able to incorporate all of the considerations listed above.

Hints

A hint is an embodiment of human/computer interface criteria for an application, stored digitally. The following example illustrates how they are used. First, suppose we have three hypothetical specific user interfaces: A, B, and C. Their style guides might specify the following:

Specific UI A

Scrolling list gadgets

Specific UI B

Requires an "Options" menu, which has "Novice" and "Advanced" menu commands. Advanced features should only appear when "Advanced" menu item is selected

Scrolling list gadgets
Pop-up list selection gadgets

Specific UI C

Intended for novice users—applications should provide basic behavior and not be overly complicated

Radio button gadgets

Example: Sample UI Component

Let's look at how we would describe a UI component with the old and the new approach. Suppose our application required the user to choose one of the 50 states. For the application, the state selection item is unimportant and unnecessary to its functionality.

Traditionally, the designer would think about the UI component and the many different ways he could present it. He would weigh the considerations imposed by the fact that the item is unimportant. If he were writing for Specific UI A, he would choose a scrolling list gadget containing the 50 states since he had no other choice. If he were writing for Specific UI B, he would have a choice between a scrolling list and a pop-up list. He would choose a pop-up list gadget containing the 50 states since it takes up less space than a scrolling list. He would also implement the functionality necessary to remove the pop-up gadget when the user chooses "novice" mode. If he were writing for Specific UI C, he would not include the option at all, since the UI is designed for novices. Each of these decisions for each of the specific UIs would be coded into each different version of the application. Changes in the UI for any one of the applications would require the program to be modified or rewritten.

Now, let's look at the new design process. Functionally, the designer knows that the user has the choice of one item out of fifty. So, he chooses a GenListClass object, which encompasses the abstract functionality of choosing from a list. He attaches a list of the 50 states to choose from. Next, he assigns subjective considerations to the object, selecting the following hints:

feature is important
feature is unnecessary to functionality of application
feature should occupy very little screen space
user does not need to see all of the options at once

That's it. He's done. No program code is written. Subsequently, when the application is executed under each of the three specific user interfaces, the associated UI interpreter chooses a gadget to fit the description.

Specific UI A Interpretation

Because the component is a list object, it is implemented as a scrolling list gadget (no other gadgets are available).

Specific UI B Interpretation

Because the feature should "occupy very little screen space, it is implemented as a pop-up list gadget. Furthermore, since it is unimportant and unnecessary", the component is removed when the user selects the Novice mode.

Specific UI C Interpretation

Because the feature is "unimportant" and "unnecessary," it is not included in the application's user interface.

So, by merely defining the application's user interface using generic UI objects and hints, the single application executable can be run under many different specific user interfaces at many different levels of functionality. To the developer, it's a better product and time and resources are saved. To the user, it's five (or more!) programs for the price of one.

Additional Hint Examples

The definition and incorporation of a comprehensive set of hint values greatly increases the adaptability of applications to future developments in user interface technology.

Therefore, the GEOS process offers many different types of hints. In addition to functional hints as described above (size, importance, etc.), there are task related hints. For instance:

This feature would appeal to someone constructing a resume
This feature would appeal to someone constructing a term paper
This feature would appeal to someone constructing a report
This feature would appeal to someone constructing a schedule

This feature would appeal to someone constructing a poster

These hints might be used by a task-oriented specific user interface. Other types and categories of hints can be defined. The more of these that are incorporated, the better the implementation of an application's user interface under future style guides (e.g. one developed for 3D holographic computer displays).

Before explaining the embodiment according to present invention, it is useful to list some more conventional definitions in a glossary.

Conventional Glossary

application

tool which allows a person to use a computer to accomplish a task

application program interface (API)

the package of the many system services that the operating system makes available to a program and the techniques developers use to call them

button

one-choice element of a control area or a menu, used in various ways such as to execute commands, display pop up windows, and display menus; See FIG. 46

check box

non-exclusive setting that shows a check mark in a square box when the setting is chosen; see FIG. 47

class

group of objects with identical types of data and methods

data structure

tables of data including structural relationships

declarative language

programing language in which the order of execution is well defined, branching and looping as necessary; individual functions and procedures operate on separately defined and maintained data

development tool

tool, generally a software program, which is either essential to the application development process or which makes the process faster and more convenient

dialog box

rectangle containing elements that elicit responses from the user, typically several at a time; the drawing of FIG. 3 shows an illustrative dialog box

environment

combination of an operating system and the particular computer on which it is used

executable

binary file which contains application code; single file which may be run by the user

graphic user interface

user interface based upon pictures and objects rather than text and commands

inheritance

classes have instance data and methods in common with classes above it in a hierarchy

instance

specific incarnation of a certain type (class) of objects

library

module of executable code which is dynamically loaded into memory when needed by one or more applications. Only one copy of a library module is loaded at a time and is shared by all executing applications

menu

rectangle containing a group of controls (basically a "multiple-choice" control). Usually accessed as a pull down menu from the main menu area or as a pop up menu from any place on the screen; the drawing of FIG. 4 shows an illustrative menu

message

an object sends a message to another object to make it perform a particular action

method

program code in an object which responds to a particular message

modal

usually used in conjunction with "dialog box'-'—means that the user must respond before continuing; he can do nothing else

object

self-contained data structure which contains instance data and methods

object oriented programming

programming language based on self-contained objects which send messages to each other to get things done

operating system

program which acts as an interface between applications and the computer hardware

pixel

single dot on the screen arranged in a rectangular grid; images on the screen are composed of many individual pixels of certain colors; the drawing of FIG. 5 shows a curve formed from individual pixels

procedural language

programming language with a well defined flow of execution during which procedures operate on data to accomplish tasks

resource

data or code, separate from actual program code, stored in a resource file

resource file

file or part of a file that contains data used by an application, such as menus, fonts, and/or icons

scalable user interface

user interface which allows the same applications to be accessed at various levels of functionality and complexity

scroll bars

control used to move the view of the data displayed in a view; the drawing of FIG. 6 shows a representative scroll bar

scrolling list

pane containing a list of text fields. The list can be read-only or it can be editable; the drawing of FIG. 7 shows a representative scrolling list

style guide

document intended to impose visual and operational consistency across the set of applications running in a particular environment. See Appendix E.

submenu

menu that displays additional choices under a menu item on a menu; the drawing of FIG. 8 shows a representative submenu

system software

see operating system

tree

hierarchy of objects; the drawing of FIG. 9 shows a sample tree hierarchy structure

user interface (UI)

set of rules and conventions by which a computer system communicates with the person operating it

user interface component

see user interface gadget

user interface gadget

item which has some function in allowing the user to communicate with the computer, e.g. button, menu, window

user interface tool kit

collection of user interface gadgets offered by an operating system for use by applications

window

rectangle containing application elements; the drawing of FIG. 10 shows a sample window

Object Oriented Programming

Object-Oriented programming is an approach to programing which is vastly different than traditional procedural programming. Programming languages such as C and Pascal consist of functions and procedures which manipulate data. The program code executes in a well-defined order, looping and branching when necessary. Object-oriented programming, on the other hand, groups data and procedures in a bundle known as an object. There is no predictable flow of execution.

The five main concepts of object-oriented programming are: objects, methods, messages, class, and inheritance. Each is described below.

Objects

Objects are self-contained units (data structures) which contain data (called instance data) and procedures (called methods) to modify their own data. Objects send and receive messages. For example, suppose a dog is an object. The commands you give him are messages. He learns those commands, and the responses he remembers are his methods. So, if you instruct the dog to "Sit," you are sending him a 637 Sit" message. He receives the "Sit" message, initiates his "Sit" method, and subsequently sits on the ground.

Messages

A message roughly corresponds to a procedure call in C or Pascal. An object sends a message to another object to make it perform a particular action. This is also known as invoking another object's method. This way of accomplishing tasks is a natural extension of how humans interact. For example, referring to FIG. 11, when a traveling salesman appears at your door, you say, "Go away," and he leaves. You give a command and expect the recipient to handle it. This is how users interact with their computers, and this is why object-oriented programming lends itself so well to a user-driven system.

Methods

Methods directly translate to procedures and functions in procedural languages. A method is the program code in the object which responds to a particular message. In the above example, the salesman's "Go away" method was his knowledge that when someone says "Go away" he should turn around, walk away, and remove your name from his list of potential customers (his instance data).

Classes

Classes are groups of objects with identical types of data and methods. Objects in a class share common sets of data and knowledge of how to respond to certain messages. Each object is called an "instance" of a class. For instance, the salesman above might be an instance of the "Acme Encyclopedia Salesmen" class. He and other fellow instances of the "Acme" class all know how to respond to a "Go away" message because of training from their supervisor.

Classes are organized in a hierarchical structure. Classes inherit behavior from classes above it. For example, the class "dog" might be defined as:

Dog
  Pretty Dog
    Poodle
    Doberman
  Ugly Dog
    Pit Bull

Sample Class Hierarchy

Inheritance

The class Dog has subclasses Pretty Dog and Ugly Dog. These subclasses may have subclasses of their own. Due to inheritance, if the class Dog contains a method "Sit," then every subclass (Pretty Dog, Ugly Dog) also understands that method. So, if an instance of the class Poodle receives a "Sit" message, it doesn't need to have its own "Sit" method. It simply passes the

message up to the class Pretty Dog which passes the message to the class Dog.

Why Is Object-Oriented Programming Natural?

The world in which we live is composed of objects. And as we saw earlier, we accomplish much of what we do by sending messages to other objects in our world and reacting to their messages. Furthermore, we generally do things by telling other objects what we want done rather than by explaining in great detail how to do them. The how describes the procedures and is part of the procedural programming model. The what describes the task, the problem, and its solution in descriptive, or declarative, terms, and is part of the declarative programming model of which object-oriented programming is a prime example. When you give your computer a print message, you don't tell it, "Now I want you to take this document that I've just finished creating and analyze its bitmap structure. Got it?" You just tell it to print and expect it to follow.

Similarly, if you give an assignment to a subordinate, you generally say, "I need the quarterly objectives report on my desk by 3:00, Jim." You don't say, "Jim, I want you to sit down at your desk. Take out a piece of paper and a pencil. Now, put at the top of the paper . . . "

But these descriptions—simplified for illustration—are good summaries of the differences between procedural programming and object-oriented programming. The world just doesn't work procedurally. Consequently it is much easier to write programs designed to emulate or simulate reality and intelligence in object-oriented programming environments than in more procedure-oriented environments. See Dan Shafer, Hyper Talk Programming.

How Is All This Implemented?

Objects are intrinsically related to classes. Think of a rolodex with printed sections for name, address, and phone numbers. Every time you fill one out, you create an object. The format of the rolodex card is the class. So then, whenever you fill out a rolodex card, you are creating an instance of the rolodex card class. The way that the format of the rolodex card is presented to an operating system is known as a data structure. Objects (and classes) are implemented as data structures. Data structures are tables of data including structural relationships. So a UI object, with its moniker, attributes, and hints would be a single data structure in the form of its class.

The GEOS Process of Application Design

Introduction

Object oriented programming is not a new concept. Neither is the idea of using objects to represent user interface components. What is novel is the way of using objects as UI components such that the GEOS operating system can interpret what those components are intended to do. Then GEOS can create the actual, visual and behavioral application UI to be a good interpretation of any number of style guides.

The GEOS process changes the process of designing an application's user interface. The user interface designer weighs the human/computer interaction design considerations against application input/output requirements and creates a generic user interface description (in the form of objects with attributes and hints). The GEOS operating system uses its specific user interface software and automated style guide interpreter to read the generic description and produce on-screen representations adhering to any particular specific user interface style guide. The GEOS system accomplishes this task through a two step process.

Firstly, the application developer defines his program's user interface using UI objects with special properties that allow him to express the user interface needs of his application.

Secondly, the GEOS system reads the description, interprets it, and produces a realization of the program user interface which visually and behaviorally conforms to the explicit and implicit guidelines of a particular style guide. Because this interpretation is done at runtime, the user may switch specific user interfaces (e.g. Motif to Openlook) at any time.

The Automated Style Guide

The GEOS process shortens and streamlines the application development process by removing the step in which the user interface is designed to fit a particular style guide. The GEOS system provides what is essentially an auto style guide. Appendix E explains the role of a style guide. Details and nuances of each specific user interface are implemented by the GEOS system. The application simply defines its user interface using a generic model. This generic model, in essence, decouples the application from its user interface. The application developer specifies the application's user interface in terms of common semantic properties, rather than specifics of the particular user interface gadgets.

As a result, the system can support a scalable environment and several GUI specifications with the same application code. Under the generic model, the developer specifies the application's user interface in terms of abstract (generic) objects, common semantic properties, and guiding hints, rather than specific user interface gadgets. These generic objects are placed in a hierarchy to demonstrate their relative importance and interdependencies.

Once the application's user interface is described in generic terms, the GEOS system maps each generic UI object to one or more specific UI objects, depending on which specific user interface is chosen. For example, an application's UI file might specify that a list of options be presented to the user. Depending on the attributes and "hints" of the generic object, this might be implemented as a submenu in OpenLook or as a dialog box in OSF/Motif™. The conversion from generic to specific user interface is transparent to the application. The GEOS system can accommodate any number of specific user interface libraries.

In this manner, the GEOS system makes sure that the end result of the specific user interface transformation conforms with its corresponding style guide. This is an important step. Style guides, as explained before, provide guidelines and specifications for application designers to follow when they design their program's user interface. Given a particular set of human-to-computer interaction needs, it defines which specific UI components to use. Sometimes style guides are very specific: for example, OpenLook specifies that main controls are to be organized in a series of button menus and most style guides ask that menu items end in an ellipses (. . .) if the user will be asked for more information before the operation is carried out. Sometimes style guides are very general: for example, there is a certain safe guide

which is not clear on, given a dialog box used to set properties, whether to supply OK, Reset, Cancel, and Help buttons or Apply, Reset, Close, and Help buttons. So, due to the complexities and nuances inherent in the process of designing the specifics of an application's user interface, developers may be forced to spend significant time and resources tweaking their designs. With the GEOS system, this step is automated and relatively painless.

This method of implementing the user interface provides benefits for developer and user alike. The user can purchase one application—a word processor, for instance. Then, depending on his personal preference, he may run the program with a Motif, OpenLook, or NewWave user interface. If his son wants to type a quick letter, he can switch to a user interface designed for novices, for example.

The developer saves untold time and resources which would have been spent designing, thinking about, and redesigning the user interface for his program. And just for one specific style guide. With the GEOS system, one application runs under different style guides (we refer to the implementation of a particular style guide as a "specific user interface.") and different levels of complexity (which is really just another style guide).

Defining an Application's User Interface

An application defines its user interface using generic UI classes.

Generic User Interface Classes

Generic UI classes are abstract types of user interface components. By thoroughly researching and analyzing existing and proposed GUIs, GeoWorks identified the major kinds of user interface components that were common. Abstracting these components—reducing them to their functional essence resulted in ten generic UI classes. For example, all specific UIs need a method of initiating an action hence the generic trigger class. A list of the major generic UI classes follows:

- GenApplication, manages the various top-level windows of an application
- GenPrimary, provides the main window for an application, grouping and managing all of the i controls and output areas for the application
- GenTrigger, represents a pushbutton that initiates a certain action when triggered by the user
- GenSummons, elicits responses from the user, typically several at a time
- GenInteraction, serves as a generic grouping object (group of controls, non-modal dialog box, menu, or sub-menu)
- GenRange, allows the user to interactively set a value within a discrete range of values
- GenList, groups multiple selection items (to set options, and so on)
- GenView, provides an area of the screen on which a document may be shown
- GenDisplay, displays and manages one or more secondary windows
- GenTextEdit and GenTextDisplay, provides text fields with differently formatted text, keyboard navigation, cut and paste, and other editing functionality

Generic User Interface Objects

Generic UI objects are instances—specific incarnations—of generic UI classes. FIG. 12 illustrates two instances, the options trigger and the enable trigger, of the GenTrigger class. When an application needs a particular UI component (a button, for instance) it chooses the appropriate generic class (GenTrigger) and asks the GEOS system to create an instance of that class. The application can then use the resulting generic UI object as part of its user interface. Each individual UI object has its own instance data whose scope is determined by the UI class. There are two kinds of instance data: attributes and hints.

User Interface Components

When an application needs a particular UI component (a button, for instance) it defines a generic UI object that represents the functionality inherent in the type of component desired. The GEOS system provides different types of generic UI objects which determine the general category of functionality wanted. Special properties of that object are set to convey more detailed as well as vague information about the human/computer interaction design considerations and application input/output requirements.

Basically, these generic UI objects are data structures with two different types of instance data—attributes and hints.

Attributes

Attributes define the behavior and/or appearance of a UI object in a very specific manner: an attribute is either on or off, and there is a definite set of attributes associated with every UI object class. When an application sets an attribute, it can be sure that the specific UI component that the GEOS system selects exhibits the desired behavior.

For example, setting the modal attribute for a dialog box ensures that the user must respond to it before continuing. Setting the disabled attribute for a trigger dims the trigger's label (called a moniker) and does not allow the user to select it.

Monikers

A moniker is a special attribute every UI object has. Each UI object may be given a moniker, or visual representation, though a moniker does need to be defined for every object. It could be the name of a button or the icon to be displayed when a window is minimized. A UI object is not restricted to a single moniker: a list of monikers may be defined. Depending on the situation and context, the GEOS system uses one of the monikers. For example, an application may define different icons for CGA, EGA, and VGA monitors to optimize the its appearance. The GEOS system displays the proper one for a particular user's set up. Some UI objects may have several textual and pictorial monikers. GEOS chooses the appropriate moniker.

Hints

Hints provide additional information about the UI object in question. An application's needs are not always absolute and may be interpreted differently (even ignored) by different specific UIs. Some visual and behavior aspects of UI objects should not be implemented as attributes because of this. In other words, there are some UI components or functionality which is not universal to all specific UIs. Those capabilities cannot be attributes, since not all specific UIs support them. Therefore, they become hints. When the developer assigns hints to a particular UI object, he cannot be

certain that the hint will be implemented by any one specific UI.

There are two types of hints: command and declarative.

Command Hints

Command hints are direct requests for a specific implementation of a UI component. A developer would choose to use a command hint when he had a specific UI component style in mind. For example, an application may explicitly ask for a scrolling list (HINT_SCROLL_LIST) or check boxes (HINT_CHECKBOXES). Not all specific UIs offer the capability to follow command hints. For instance, some specific UIs allow the user to use the keyboard to navigate menus and dialog boxes. To support this, certain UI objects would contain several HINT_NAVIGATION_ID and HINT_NAVIGATION_NEXT_ID hints. For example Motif might make use of this hint, but OpenLook might ignore it because the style guide does not allow such navigation. The GEOS system fulfills a particular command hint in any specific user interface that supports it.

Declarative Hints

Declarative hints are more vague; without referring specifically to a particular implementation, they give an indication of the functionality of the UI object in question. For example, a generic UI object containing a list of possible actions may have a HINT_MENUABLE, indicating that the developer envisions the list being presented in a menu. However, perhaps a specific UI designed for novice users states that a menu is too complex. Then the GEOS system implements the list of actions as a simple series of large, plainly visible buttons. Or, similarly, an option in that menu may have hints stating that it is advanced, infrequently used, and potentially dangerous. Then a novice specific UI would remove the trigger altogether.

Once again, declarative hints may or may not be implemented by a particular specific UI. For instance, CUA does not allow submenus in the menu bar. A GenInteraction object with the hint HINT_MENUABLE that is inside of another GenInteraction object with the hint HINTMENUABLE would be implemented as a submenu in OpenLook or Motif. However, in CUA, it would be added to the menu and set apart by separators, since submenus are illegal according to the style guide.

Using Generic UI Objects

The generic UI objects an application chooses to represent the UI components it needs are arranged into a tree. This tree is a hierarchy of UI objects, to convey the relative importance and interdependencies of each object. This provides an indication of which components ought to be in plain view and which can be hidden one or more layers deep. The illustrative drawings of FIG. 13 show an example of such a generic UI tree. A description of the generic UI tree of FIG. 13 is provided in Appendix A. Appendices A-E are expressly incorporated herein by this reference.

Given a generic user interface description such as the one in FIG. 13, the GEOS system can implement it in a number of different specific user interfaces. It automatically sizes menus, fields, and boxes; it places buttons, scroll bars, and text—all the while adhering to the specific user interface style guide. The illustrative drawing of FIG. 14 shows how this particular generic UI specification might be realized by GEOS for Motif. Note that GenApplication has no visual representation.

Decorations

Decorations are additional specific user interface components that the developer does not request, but that the GEOS system provides in order to maintain a good implementation of a particular style guide. For example, note that in the sample application above, GEOS adds the buttons in the upper corners, a resizing border, and a "pin" option in the menu. These are all accoutrements which the Motif style guide, for example, states should exist and function in a certain manner. The developer does not need to worry about remembering them or asking for them, since they may be different for Open Look or New Wave, for example. This is another example of how the GEOS system ensures a good interpretation of style guide without needing explicit direction from the programmer.

Specific User Interfaces

Specific user interfaces are implemented as libraries. Much like a group of students can go to the public library and all share an encyclopedia, programs can share library modules. Libraries are modules of executable code which are dynamically loaded into memory when needed by one or more applications. Only one copy of a library module is loaded at a time and is shared by all executing applications. Specific UI libraries are responsible for interpreting the generic UI description and implementing the actual application's user interface.

Programming Examples

To ground out the above concepts, let's compare designing a simple user interface the conventional way versus the GEOS way. We will not worry about the underlying application functionality. We shall create a simple user interface in two different specific exemplary user interfaces (Macintosh and OS/2 Presentation Manager). We will end up with two separate executable applications. Then we shall do the same in accordance with the GEOS process, and show how the resulting single executable application can be displayed in any number of specific user interfaces.

Macintosh Example

Let's create a simple user interface on an Apple Macintosh—a single window with a File menu containing five commands, New, Open, Save, Save As, and Quit.

Macintosh applications make use of many resources, such as menus, fonts, dialog boxes, and icons, which are stored in resource files. For example, an icon resides in a resource file as a 32-by-32 bit image, and a font as a large bit image containing the characters of the font. In some cases the resource consists of descriptive information (such as, for a menu, the menu title, the text of each command in the menu, whether the command is checked with a check mark, and so on). The resources used by an application are created and changed separately from the application's code. This separation is the main advantage to having resource files. A change in the title of a menu, for example, will not require any recompilation of code, nor will translation to another language. The preceding together with the following

description and code fragments are from, Inside Macintosh, Volume 1.

So, to create the sample application, the programmer would first make use of graphical, interactive development tools on the Macintosh to define the menu and its contents. He would first create a new menu resource. Then he would add commands to the menu (New, Open, Save, Save As, and Quit). Finally, he sets the attributes of the menu and its choices (e.g. no checkmarks, separators between Save As and Quit, etc.). Below is a complete list of all the resources he would define:

- Menu (resource ID #128)-menu with the apple symbol as its title and no commands in it
- Menu (resource ID #129)-file menu with commands New, Open . . . , Save, Save As . . . , and Quit commands
- Window template (resource ID #128)—document window without a size box; top left corner of (50,40) on coordinate plane, bottom right corner of (300,450); title "Sample"; no close box

Each menu resource also contains a "menu ID" that's used to identify the menu when the user chooses a command from it; for all three menus, this ID is the same as the resource ID.

Excerpts of code to initialize and display these resources are provided in Appendix B.

The code in Appendix B and the resource file would result in the on-screen shown in FIG. 15.

Note that the code required to create this particular on-screen representation is very specific to the Macintosh. For instance, if you would like to make the application's appearance and behavior conform to the CUA (Sort Presentation Manager) style guide, everything would have to be rewritten.

OS/2 Presentation Manager Example

Let's create the same user interface in OS/2 Presentation Manager. OS/2's style guide (CUA) and operating system are quite different than Apple's, so user interface designs need to be altered and code needs to be completely rewritten.

To create an application's user interface in OS/2 Presentation Manager, the code describing the user interface is partially imbedded in the actual program code. To display standard window with a simple menu, the designer would include the lines shown in Appendix C in his main program file (e.g. SAMPLE.C). The code fragments in Appendix C are from Programming the OS/2 Presentation Manager by Charles Petzold.

The code and resource files of Appendix C are compiled and the resulting on-screen representation of the application would look similar to the representation of FIG. 16.

Notice that when developing in either OS/2 Presentation Manager or Apple Macintosh, the programmer defines specific user interface components with specific attributes. Then the program code accesses them and the operating system draws them on the screen. The actual mechanics of programming and development in these two environments are very different.

In Presentation Manager, menus and menu-related attributes are defined in a textual resource file. Attributes of windows and other UI components are defined via routine calls. Options are passed as parameters.

On the Macintosh, UI components are contained in separate resource files. Thus, their attributes, are defined using a resource editor application. This application supplies graphical templates which the programmer uses to create and edit resources. For example, the programmer formats dialog boxes in the editor, manually sizing the border, adding text blocks, setting text styles, placing buttons, and so forth.

GEOS Example

In the GEOS process, the programmer defines the user interface needs of his application with generic UI objects. These objects have attributes, as do the OS/2 or Macintosh objects, but these attributes are only those that represent an aspect of apace or behavior which is common to all specific UIs. Peculiarities of different specific UI implementations are accommodated through the use of hints.

Appendix D provides a sample user interface description file.

User Interaction

Perhaps the most vital aspect of computer use is user interaction with the computer, for what use are hundreds of features and a clear and concise user interface, if the user cannot easily make use of them? Most graphically oriented systems make use of a device called a mouse. The user slides the palm-sized object around on his desk, and a pointer on the screen moves in accordance. A mouse may have one to three buttons. Moving the pointer around the screen, the user can click on buttons, resize windows, and draw circles. However, just as nearly every specific user interface has a style guide describing visual and behavioral aspects of the system, nearly every specific user interface style guide defines user interaction conventions. There are so many ways that a user could possibly click and drag and double click the left button and triple click the right button and so on that these conventions are necessary.

So, once again, Geoworks identified a problem. Conventional applications generally handle their own user interaction. For example, referring to FIG. 17, if the user double clicks on any letter in a word, the application selects the entire word because that is what its style guide says to do. Notice that we have a parallel without our earlier dilemma—different style guides have different ways of handling user input. How can an application be truly specific user interface independent if it has to worry about different types of user input? The answer according to the present invention is to abstract user interaction as well.

Let's follow how user interaction works in the GEOS system with reference to FIG. 18. First, the application receives user input. For instance, the user double clicks while using a word processor under Motif. Then, the application determines the context of the user input. For instance, the user clicked on the second word in a word processing document. Next, the application passes this information, the actual input and the context, to the appropriate specific UI interpreter. Finally, the UI interpreter, given the context and the raw input, tells the application exactly what to do. For example, it tells the word processor to select the targeted word.

Conventional Development

There are two major problems with conventional user interface development: the time involved and the potential for difficult to use application user interfaces.

Time

The current state-the-art operating systems offer a variety of tools and utilities to make the developer's life easier. However, no matter whether the program's environment is the NeXT computer or Microsoft Windows for the PC, for example, the developer still has to manually lay out every user interface component. He chooses the window style. He places the menu items. He adds buttons and dialog boxes. He carefully places list of options and text fields in those dialog boxes, perhaps moving them around a pixel (a single dot on the screen) at time. He defines the exact sizes and locations of every user interface component. Then he steps back and makes sure that the resulting interface still adheres to the style guide set forth for his environment. He tweaks the design some more, steps back, and tweaks it some more. Finally, when the user interface design is finally completed, the programming team write the rest of the program. Generally, at least 30 percent of development time is spent designing and implementing the user interface.

It is a time-consuming task to redesign the user interface (that took so long to complete) of an application for other environments. In some environments it is easier than others. But in all of them, the designer is constantly adjusting and worrying about making the result conform with the style guide. And that can require much time and hard hard work. Modifying it to adhere to another style guide can also take much time and hard work.

Differences in the Quality of UIS

There are many very good applications with intuitive and logical user interfaces. There are also a lot of very powerful applications with user interfaces that are not as intuitively easy to use. When the developers are solely responsible for making sure that their applications correspond with the goals and objectives set forth in a style guide, there are bound to be some odd interpretations. Given a stack of wood and power tools, a master carpenter could build a beautiful and priceless bird house. A less skillful craftsman might build a worthless doorstop. Similarly, given the tools for creating a user interface, programmers could very easily create less than optimal user interfaces. For example, examine the case of a dialog box summoned when a user selects the print command. There are many, many different ways that an application could handle this situation. Some are satisfactory, some are excellent, and others are less than optimum. For instance, FIG. 19 shows a good layout for the print dialog box. The dialog box design of FIG. 19 is good for several reasons. Firstly, it visually groups options into logical groups with sensible titles—Printer Options and Document Options. The setting of which printer is connected is not one which the user frequently changes. Therefore, the options related to this are not even accessed through this dialog box. Clicking Change Options . . . brings up a separate dialog box. The Document Options have descriptive, obvious names—high, medium, and low print quality. Print and Cancel give an good indication of what the buttons will do. Additionally, the extra box around Print indicates a default action, good for experienced users as well as novices unsure of what to do next.

In contrast, the dialog box of FIG. 20 is can be a challenge to use. The options are not obviously grouped into logical divisions. The title Other Options is somewhat vague. "Configuration" is a technical term. Printer Configuration options are not often accessed, and possibly should not be here, for example. Clicking to cycle through the choices is not an intuitive or friendly way to accomplish the task-it doesn't plainly show the user all the possible choices, and if the user passes the proper setting, he will have to keep clicking to get back to it. Print Quality options use is unclear; technical terms—NLQ, regular, and draft. What does mean? Is regular latter looking than NLQ? Go and Stop buttons are unclear as to their function. There is no default action.

And on top of it all, once the developers have created a good interface for their application and spent lots of time and money doing it, they still have to create an entirely view executable program to run in a different environment under a different specific UI.

Scalable User Interface

The scalable user interface can be thought of as just another style guide. It is simply a style guide designed with the user's computer proficiency very much in mind. For example, for novices, the style guide would state that the user should be able to plainly see all his options. Thus, hidden menus (pulldown or pop up) would not be allowed. Scrolling views are also undesirable because of their complexity. Visible methods of getting help need to be evident at all times. Thus, an automated style guide can run an application with its normal style guide (such as Motif or Open look), or switch to one designed for novices. Conceptually and to the application developer, it's just the same as switching between two very similar "professional" specific user interfaces. To the user, it's like getting several programs for the price of one.

Operation

Referring to FIG. 21, there is shown a diagram illustrating the dynamic interaction of the constituent elements of the invention. The elements of FIG. 21 all are implemented in computer software. The Application software interacts with the operating system software. The operating system software includes the Generic User Interface Object Library and Controller (GUIOLC), multiple specific UI Interpreters (SUIIs) (only one shown), and multiple specific UI Toolbox and Controllers (SUITC) (only one shown), and their respective driver software modules (only one set of driver software modules shown).

Application Data is operated upon by the Application Software. A Generic UI Specification (GUIS), which is associated with the Application, is operated upon by the GUIOLC. Specific UI Application Interface Data is operated upon by the SUITC.

Multiple Applications can run simultaneously. Each Application corresponds to a particular GUIS. It is possible to have multiple GUISs, that is, different GUISs for the different Applications.

The GUIOLC and the SUII serve to map Input/Output (I/O) requirements of an Application to the SUITC under which the Application is to be presented to the user. In the present embodiment, the GUIOLC provides a series of generic UI object classes (e.g., GenApplication, GenPrimary, GenTrigger, etc. . . .). These generic UI classes act as an interface between the Application and the portion of the operating system software that controls the representation of a specific user interface for the Application. For example, when the Appli-

cation needs to represent a UI component used to initiate a certain user action, it specifies the GenTrigger generic user interface object. From the standpoint of the Application, the steps required to represent a component for initiating user interaction merely involves specifying the GenTrigger object. As explained below, the operating system software, in accordance with the invention, handles the details of actually selecting, arranging and otherwise managing the gadgets used to represent the component.

Once the Application has specified a particular generic user interface object, a selected SUII uses the specified object and instance data for that object to interpret the manner in which the specified object is to be represented. In particular, the selected SUII selects gadgets from a corresponding SUITC and arranges the gadgets in accordance with attributes and hints in the instance data for the specified object.

Each Application can have a different GUIS associated with it. Thus, while two applications might specify the same generic UI object, the different GUISs associated with the different Applications can result in different representations (visual or behavioral) for the same generic UI object. This is because different GUISs can have different instance data.

In the present invention, the Operating System Software, rather than the application, indicates the specific UI under which the Application is run. Thus, for example, if there are four possible specific UIs (with four corresponding SUISs and four corresponding SUITCs) then the system software determines which of the four is to be used by the Application (and which of the four SUIIs and SUITCs). However, it is possible for the application itself to indicate which of the four (or more) specific UIs is to be used by the Application.

FIGS. 23-26 illustrate how the same generic UI object and the GUIS for a particular Application can result in different visual representations when different specific UIs are designated. In FIG. 23, a GenDocumentControl object and the instance data from an Application GUIS is shown. FIG. 24 shows a possible NewWave interpretation of the object of FIG. 23. FIG. 25 shows a possible OpenLook interpretation of the object of FIG. 23. FIG. 26 shows a possible Motiff interpretation of the object of FIG. 23.

FIGS. 27-30 further represent how the same generic UI object and a GUIS for a particular Application can result in different visual representations when different specific UIs are designated. FIGS. 28-30 respectively represent possible NewWave, OpenLook and Motiff interpretations of the object of FIG. 27.

FIG. 31 illustrates the operation of a representative SUII for a GenList object under an OpenLook User interface. Possible gadget choices available from the corresponding OpenLook SUITC are indicated in the left column. The representation and arrangement of the gadgets in accordance with this SUII is indicated in the center column. The decision method used to determine which gadget choice to make is indicated in the right column.

Thus, the representative interpreter selects which gadgets (left column) and their arrangement (center column) based upon predetermined criteria (right column). The information used to test the criteria is found in the instance data of the GUIS for the designated GenList object.

It should be noted that, for example, one Application may specify certain instance data in its GUIS for the GenList object, and another Application might specify different instance data in its GUIS for the GenList object. The operating system using the exemplary SUII of FIG. 31, therefore, could represent a GenList object differently for the two Applications due to their different GUISs.

FIG. 32 illustrates the operation of another exemplary SUII. The interpreter of FIG. 32 is hypothetical for a GenList object. The left column represents possible gadgets from the hypothetical SUITC (not shown). The center column represents their arrangements under this interpreter. The right column illustrates the criteria used to select the gadgets.

From FIGS. 31 and 32, it should be appreciated that even the same generic UI object (e.g. GenList) using the same instance data from the same GUIS can result in a different UI representation when a different specific UI is selected. For example, in FIG. 31, the specific UI is OpenLook, and in FIG. 32, the specific UI is a hypothetical UI.

FIG. 33 shows a further representation of a generic user interface object (GenList) and its instance data and two possible interpretations of it, one under a hypothetical UI, and the other under an OpenLook UI.

FIG. 34 shows a representative hierarchy of generic UI objects and their respective instance data. FIG. 35 shows possible Motiff and OpenLook interpretations. FIG. 36 shows a possible hypothetical graphical UI interpretation. FIGS. 37 and 38 respectively show possible hypothetical interpretations under advanced and novice modes.

FIG. 43 provides a dynamic block diagram which represents interpretation of user interaction by an operating system in accordance with the present invention. A user provides an input such as a double click mouse command on text. The Application passes the user input command information (double click) and the context information (over text) to a specific UI interpreter. The SUII interprets the input information and indicates its meaning to the Application. The Application then can request the operating system to perform a function consistent with the input (e.g., select a targeted word).

It will be understood that different specific UIs can interpret the same input differently. Moreover, the different interpretations can depend not only upon the nature of the command but also upon the context in which the command is provided. The SUII shields the Application from the details of user input interpretation. Of course, as explained above, there may be multiple specific UIs supported by the operating system. As explained above, the different SUIIs for the different specific UIs may interpret the user input (command plus context) differently.

Referring to FIG. 41, there is provided a dynamic block diagram which provides a generalized representation of the operation of an object oriented system. The present invention is implemented as an object oriented system, although it could be implemented as a procedural system (FIG. 39).

In the presently preferred embodiment, each generic UI object represents a class. The GUIS for an Application provides instance data for the generic UI object class members. The multiple SUIIs include messages that point to methods for operating on the instance data.

Thus, for example, when an Application is running under a first specific UI, the generic UI object points to the SUII for the first specific UI, and the messages and methods of that first SUII operate on the instance data

of the generic UI object. If, on the other hand, an Application is running under a second specific UI, the generic UI points to the SUII for the second specific UI, and the messages and methods of the second SUII operate on the instance data of the generic UI object.

It will be appreciated, for example, that an Application and an SUII can communicate through a generic UI object. For example, referring to FIG. 31, the Application may specify the generic UI object GenList and communicate the message, delete "tomato". The GenList object, running under the OpenLook specific UI, for example, sends the message to the OpenLook SUII. The OpenLook SUII uses the message to identify a method that results in removal of the "tomato" moniker from the UI representation.

Style Guides

Style guides are documents intended to promote both visual and operational consistency across the set of applications running in a particular environment. To achieve this goal, design rules describe the user interface and design approach in detail. However, it is impossible to anticipate all situations. So that consistent extensions can be made, portions of the document attempt to explain the rationale behind the rules, and the intended "feel" of the applications in question. These design rules are provided in pursuit of integration and consistency. Application programmers are asked to commit themselves to following the design rules because of the importance of a cohesive, consistent set of applications. See, for example, HP NewWave Environment: User Interface Design Rules.

For example, the following is an excerpt from *Open Look Graphical User Interface Application Style Guidelines.* It describes behavioral and visual guidelines for scroll bars (which allow the user to view portions of a large document at a time by "scrolling" up and down and left and right).

"Scrolling with Scrollbars: This section describes information you need to specify for your application when you provide scrollbars for a scrollable text region . . ."

"Scrolling Objects of Unknown Size: In some situations, it is impossible to determine the size of the object being viewed. For example, the result of a database query might be read in only as needed. Such situations call for a slight modification of the usual scrollbar behavior.

When the size of the entire object is not known, make the length of the proportion indicator represent the length of the part of the object that is known at any given point."

"If users scroll to the end of the cable—either by dragging the elevator or by clicking on the end cable anchor—scroll the view to the end of the data that has already been read in. To leave the elevator at the very end would be misleading, because the view is not at the end of all the data.

When the elevator is not at the end of the data, bump the elevator a few pixels upward from the bottom cable anchor to show that the view is not at the true end of the data. Put a message in the footer of the window to inform users about what is happening.

When users drag the elevator again or click on the down (or right) arrow, interpret that action as a signal that users want to read in the next portion of the data."

"Once the new data is read in, the scrollable object is larger, and you will need to adjust the position of the elevator accordingly."

Releasing the application developer from having to deal with pages and pages of this is what the patent is all about.

While a particular embodiment of the invention is shown and described, it will be appreciated that the present system can be implemented differently without departing from the invention. For example, as illustrated in FIG. 39, the invention can be implemented as a procedural system rather than as an object oriented system. Moreover, for example, in the present embodiment, the GUIOLC and the multiple SUIIs are separate modules. The GUIOLC and the multiple SUIIs can be implemented as a single module without departing from the invention.

Appendix A

```
/* ------------------------------------------------------------
      Application Object
------------------------------------------------------------*/
/* This application is in its own resource so that geoManager
 * can load it quickly to grab the icon for the application. */

start     AppResource

SampleApp = GenApplication {
          /* GeoManager uses this token information to store the
           * application's icon in a database. */
      tokenChars = "'S','A','H','P'";
      tokenID = "MANUFACTURER_ID_GW"
      children = SamplePrimary;                  /* one child */
      active = YodaPrimary;                            /* have window
                                                 appear when launched. */

}

end AppResource;
```

```
/* ------------------------------------------------------------
     Primary Window
------------------------------------------------------------*/

start Interface;       /* This resources holds misc UI objects. */

SamplePrimary = GenPrimary {
      moniker = "Sample Application";
      genStates = default -maximized;  /* Do not open */
      children = SampleView, SampleMenu  /* maximized */
      hints = {
                HINT_NOT_MINIMIZABLE
      }
}

end Interface;

/* ------------------------------------------------------------
     UI Objects within Primary Window
------------------------------------------------------------*/

start Interface;       /* This resources holds misc UI objects. */

SampleView = GenView {
      viewAttributes = isolatedContents, grabWhilePressed,
                       dragScrollingOn;
      output = process;    /* send exposed method to appl */
      backColorR = BLACK  /* background color */
      horizOpenSize = 256;
      vertOpenSize = 256;
      horizAttributes = scrollable;
        vertAttributes = scrollable;
  }

  end Interface;

  /* ------------------------------------------------------------
       Menus
  ------------------------------------------------------------*/

  start MenuResource;

  SampleMenu = GenInteraction {
        moniker = "Interaction";
        hints = {
                  HINT_MENUABLE     /* all of the children */
                                    /* can be placed in a menu */
        }
        children = MenuItem1, MenuItem2;
  }

  MenuItem1 = GenTrigger {
        moniker = "Trigger 1";
  }
```

```
    MenuItem2 = GenTrigger {
          moniker = "Trigger 2";
    }

    end MenuResource;
```

Appendix B

```
- - -
{The USES clause brings in the units containing the Pascal }
{interfaces. The SU expression tells the compiler what file to look }
{in for the specified unit. }
USES {SU Obj/MemTypes } MemTypes,   {basic Memory Manager data types}
     {SU Obj/QuickDraw} QuickDraw,  {interface to QuickDraw}
     {SU Obj/OSIntf   } OSIntf,     {interface to the Oper System}
     {SU Obj/ToolIntf } ToolIntf;   {interface to the Toolbox}

CONST appleID = 128;   {resource IDs/menu IDs for Apple, File menu}
      fileID  = 129;

      appleM = 1;      {index for each menu in myMenus (array of }
      fileM  = 2;      {menu handles)}

      menuCount = 2;   {total number of menus}

      windowID = 128;  {resource ID for application's window}

      newCommand    = 1; {menu item numbers identifying commands in}
      openCommand   = 2; {File menu}
      saveCommand   = 3;
      saveAsCommand = 4;
      exitCommand   = 6; {skip a number because of separator}

VAR myMenus: ARRAY[1..menuCount] OF MenuHandle;
      { array of handles to the menus }
    wRecord: WindowRecord;  {info about the application window}
    myWindow: WindowPtr;    {pointer to wRecord}
- - -

PROCEDURE SetUpMenus;
{ Set up menus and menu bar }

  VAR i:INTEGER;

BEGIN
{Read menu descriptions from resource file into memory and store }
{handles in myMenus array }
 myMenus[appleM] := GetMenu(appleID);
   { read Apple menu from resource file }
 AddResMenu(myMenus[appleM],'DRVR');
   { add desk accessory names to Apple menu }
```

```
 myMenus[fileM] := GetMenu(fileID);
   { read File menu from resource file }

 FOR i:=1 TO menuCount DO InsertMenu(myMenus[i],0); {install menus}
  DrawMenuBar;                          {in menu bar and draw menu bar}

END;  { of SetUpMenus }
- - -

BEGIN   {main program}
{ Initialization }
- - -
InitWindows;          {initialize Window Manager}
InitMenus;            {initialize Menu Manager}
TEInit;               {initialize TextEdit}
InitDialogs(NIL);     {initialize Dialog Manager}
InitCursor;           {call QuickDraw to make cursor an arrow}

SetUpMenus;           {set up menus and menu bar}
- - -
myWindow := GetNewWindow(windowID,@wRecord,POINTER(-1));
  { put up application window }
SetPort(myWindow);
  { call QuickDraw to set current grafPort to this window }
- - -
END.
```

Appendix C

```
- - -
CHAR szClientClass [] = "Sample" ;
HAB  hab ;

int main (void)
      {
      static ULONG flFrameFlags = FCF_TITLEBAR | FCF_SYSMENU |

                           FCF_SIZEBORDER | FCF_MINMAX |
                           FCF_SHELLPOSITION |
                           FCF_TASKLIST | FCF_MENU ;
HMQ   hmq ;
HWND  hwndFrame, hwndClient ;
QMSG  qmsg ;

hab = WinInitialize (0) ;
hmq = WinCreateMsgQueue (hab,0) ;

WinRegisterClass (hab,            // Anchor block handle
                  szClientClass,  // Name of class being
                                  //  registered
                  ClientWndProc,  // Window procedure for class
                  0L,             // Class style
                  0) ;            // Extra bytes to reserve
```

```
hwndFrame = WinCreateStdWindow (
              HWND_DESKTOP,     // Parent window handle
              WS_VISIBLE,       // Style of frame window
              &flFrameFlags,    // Point to control data
              szClientClass,    // Client window class name
              NULL,             // Title bar text
              0L,               // Style of client window
              NULL,             // Module handle for resources
              ID_RESOURCE,      // ID of resources
              &hwndClient) ;    // Pointer to client window hndl

winSendMsg (hwndFrame, WM_SETICON,
            WinQuerySysPointer (HWND_DESKTOP, SPTR_APPICON,
            FALSE),
            NULL) ;      // Set minimized icon for window
```

This code creates a standard application window with attributes defined by **flFrameFlags.**

- FCF_TITLEBAR, creates a title bar
- FCF_SYSMENU, creates a system menu
- FCF_SIZE BORDER, creates a sizing border
- FCF_MINMAX, creates a minimize and maximize button
- FCF_SHELLPOSITION, the Presentation Manager (shell) determines the position of the window, typically in a cascaded position from the last application that started
- FCF_TASKLIST, adds the window to the switch list of the Task Manager
- FCF_MENU, creates a menu bar

The menu options are defined in a separate resources file (SAMPLE.RC). Note that the CUA style guide specifies a different order of commands as well as "exit" rather than "quit"

```
/* ------------------------------
   SAMPLE.RC resource script file
   ------------------------------*/

#include <os2.h>
#include "sample.h"

MENU ID_RESOURCE
     {
     SUBMENU "~File",        IDM_FILE
     MENUITEM "~Open...",            IDM_OPEN
     MENUITEM "~New",                IDM_NEW
     MENUITEM SEPARATOR
     MENUITEM "~Save",               IDM_SAVE
     MENUITEM "Save ~As..."          IDM_SAVEAS
     MENUITEM SEPARATOR
     MENUITEM "E~xit",               IDM_EXIT
     }
```

Appendix D

```
#include "generic.uih"

/* ------------------------------------------------------------------
     Application Object
------------------------------------------------------------------*/
/* This application is in its own resource so that geoManager

       )
       children = FileSubMenu,
                  ExitGroup;
 )

 FileSubMenu = GenInteraction {
       children = NewTrigger,
                  OpenTrigger,
                  SaveTrigger,
                  SaveAsTrigger;
       hints = {
                HINT_MENUABLE,          */ can be put in menu */
                HINT_SUB_GROUP          */ add separator */
       }
 }

 NewTrigger = GenTrigger {
      NEW_TRIGGER_VALUES                */ moniker = 'N', "New"; */
 }

OpenTrigger = GenTrigger {
      OPEN_TRIGGER_VALUES              */ moniker = 'O', "Open"; */
      hints = {
              HINT_BRINGS_UP_WINDOW    */ append "_" */
      }
]

SaveTrigger = GenTrigger {
      SAVE_TRIGGER_VALUES              */ moniker = 'S', "Save"; */
}

SaveAsTrigger = GenTrigger {
      SAVEAS_TRIGGER_VALUES            */ moniker = 'A', "Save As"; */
      hints = {
              HINT_BRINGS_UP_WINDOW
      }
}
```

```
FileExit = GenTrigger {
        EXIT_TRIGGER_VALUES     */ kbdAccelerator = specificUI F3; \
                                   moniker = 'E', "Exit"; */
}
end MenuResource;
```

What is claimed is:

1. A method for invoking a user interface for use with an application operating in a computer system comprising the steps of:

providing in the computer system a generic object class that corresponds to a class of function performed with the user interface;

specifying in the application instance data in the form of a generic object specification that corresponds to the generic object class, the instance data including attribute criteria, which are criteria that must be met by a specific user interface implementation that is selected using the instance data, and hint criteria, which are criteria that are permitted but not required to be specified in the instance data, and if specified, are permitted but not required to be met by a specific user interface implementation that is selected using the instance data;

providing in the computer system a specific use interface toolbox and controller that operates in the computer system to provide a selection of possible specific user interface implementations for use in performing the class of function; and

providing in the computer system an interpreter for the specific user interface toolbox and controller, the interpreter operating in the computer system to select a specific user interface implementation from the selection of possible specific user interface implementations, such that a selected specific user interface implementation satisfies both the attribute criteria and hint criteria specified for the generic object class, except if no specific user interface implementation satisfies both the attribute criteria and hint criteria specified for the generic object class then the interpreter being operable to select another specific user interface implementation that satisfies the attribute criteria but not all of the hint criteria that have been specified for the generic object class.

2. The method of claim 1 wherein said step of providing a generic object class in the computer system includes providing a library of generic objects classes, each respective generic object class in said library representing a respective class of functions.

3. The method of claim 1 wherein:

said step of providing a generic object class in the computer system includes providing in the computer system multiple generic object classes, each of which corresponds to a different class of function performed with the user interface; and

said step of specifying instance data includes specifying in the application multiple respective generic object specifications.

4. The method of claim 3 wherein said step of specifying multiple respective generic object specifications includes specifying a tree hierarchy relationship among the multiple respective generic object classes.

5. The method of claim 3 wherein:

said step of specifying multiple respective generic object specifications includes specifying a tree hierarchy relationship among the multiple respective generic object classes; and

said tree hierarchy provides an indication of which visual user interface components are to be in plain view and which such visual user interface components are to be hiddend by other such components.

6. The method of claim 1 wherein the generic object class is of a GenListClass that encompasses the generic user interface functionality of selecting from among multiple choices.

7. The method of claim 1 wherein the generic object class is of a GenTriggerClass that encompasses the generic user interface functionality of using a screen image to invoke an action.

8. A method for invoking a user interface for use with an application operating in a computer system comprising the steps of:

providing in the computer system multiple respective generic object classes that respectively correspond to respective classes of function performed with the user interface;

specifying in the application instance data in the form of a generic object specification that corresponds to a designated one of the respective generic object classes provided in the computer system, the instance data including respective attribute criteria, which are criteria that must be met by a specific user interface implementation that is selected using the instance data nd respective hint criteria, which are criteria that are permitted but not required to be specified in the instance data, and if specified, are permitted but not required to be met by a specific user interface implementation that is selected using the instance data; and

providing in the computer system a specific user interface toolbox and controller that operates in the computer system to provide a selection of possible specific user interface implementations for use in performing the class of function of the designated generic object class; and

providing in the computer system an interpreter that corresponds to the specific user interface toolbox and controller, the interpreter operating in the computer system to select a specific user interface implementation from the selection of possible specific user interface implementations, such that a selected specific user interface implementation satisfies both the respective attribute criteria and the respective hint criteria specified for the designated generic object class, except if no specific user interface implementation satisfies both the respective attribute criteria and the respective hint criteria specified for the designated generic object class then the interpreter being operable to select another specific user interface implementation that satisfies the respective attribute criteria but not all of the hint criteria that have been specified for the designated generic object class.

9. A method for designating a user interface for use with an application operating in a computer system comprising the steps of:

providing in the computer system a first generic object class and a second generic object class wherein each such respective generic object class corresponds to a respective class of function to be performed using the user interface;

specifying in the application first instance data in the form of a first generic object specification that corresponds to the first generic object class, the first instance data including first attribute criteria, which are criteria that must be met by a specific user interface implementation that is selected using the first instance data, and first hint criteria, which are criteria that are permitted but not required to be specified in the first instance data and if specified, are permitted but not required to be met by a specified user interface implementation that is selected using the first instance data;

specifying in the application second instance data in the form of a second generic object specification that corresponds to the second generic object class, the second instance data including second attribute criteria, which are criteria that must be met by a specific user interface implementation that is selected using the second instance data and second hint criteria, which are criteria that are permitted but not required to be specified in the second instance data, and if specified, are permitted but not required to be met by a specific user interface implementation that is selected using the second instance data;

providing in the computer system a specific user interface toolbox and controller that operates in the computer system to provide, for each of the first and second generic object classes, a respective selection of multiple possible specific user interface implementations;

providing in the computer system an interpreter that corresponds to the specific user interface toolbox and controller, the interpreter including a first generic object class interpreter and a second generic object class interpreter;

producing a first specific user interface interpretation for the first generic object class from the specific user interface toolbox and controller using the first generic object class interpreter, such that the first specific user interface interpretation satisfies both the first attribute criteria and the first hint criteria, except if no first specific user interface interpretation satisfies both the first attribute criteria and the first hint criteria, then using the first generic object class interpreter to produce another first specific user interface interpretation that satisfies the first attribute criteria but not all of the first hint criteria that have been specified; and

producing a second specific user interface interpretation for the second generic object class from the specific user interface toolbox and controller using the second generic object class interpreter, such that the selected second specific user interface interpretation satisfies both the second attribute criteria and the second hint criteria, except if no second specific user interface interpretation satisfies both the second attribute criteria and the second hint criteria, then using the second generic object interpreter to select another second specific user interface interpretation that satisfies the second attribute criteria but not all of the second hint criteria that have been specified.

10. A method for designating a user interface for use with an application operating in a computer system comprising the steps of:

providing in the computer system a generic object class that corresponds to a class of function that is to be performed with the user interface;

specifying in the application instance data that corresponds to the generic object class, the instance data including attribute criteria, which are criteria that must be met by a specific user interface implementation that is selected using the instance data, and hint criteria, which are criteria that are permitted but not required to be specified in the instance data, and if specified, are permitted but not required to be met by a specific user interface implementation that is selected using the instance data;

providing in the computer system a first specific user interface toolbox and controller that is operable in the computer system to provide a first selection of multiple possible first specific user interface implementations for use in performing the class of function;

providing in the computer system a second specific user interface toolbox and controller that is operable in the computer system to provide a second selection of multiple possible second specific user interface implementations for use in performing the class of function;

providing in the computer system a first interpreter that corresponds to the first specific user interface toolbox and controller, the first interpreter operable in the computer system to select a first specific user interface implementation from the first selection of possible first specific user interface implementations, such that a selected first specific user interface implementation from the first selection satisfies both the attribute criteria and the hint criteria specified for the generic object class, except if no first specific user interface implementation from the first selection satisfies both the attribute criteria and hint criteria specified for the generic object class then the first interpreter being operable to select another first specific user interface implementation from the first selection that satisfies the attribute criteria but not all of the hint criteria that have been specified for the generic object class;

providing in the computer system a second interpreter that corresponds to the second specific user interface toolbox and controller, the second interpreter operable in the computer system to select a second specific user interface implementation from the second selection if possible second specific user interface implementations, such that a selected second specific user interface implementation from the second selection satisfies both the attribute criteria and the hint criteria specified for the generic class, except if no second specific user interface implementation from the second selection satisfies both the attribute criteria and hint criteria specified for the generic object class then the second interpreter being operable to select another second specific user interface implementation from the second selection that satisfies the attribute criteria but not all of the hint criteria that have been specified for the generic object class;

selecting one of the first and second specific user interface toolbox and controllers and a corresponding one of the first and second interpreters; and

producing one of a first specific user interface implementation and a second specific user interface implementation using the selected controller and the selected interpreter.

11. A method for designating a user interface for use with an application operating in a computer system comprising the steps of:

A. providing in the computer system a first generic object class that corresponds to a first class of function and a second generic object class that corresponds to a second class of function;

B. specifying in the application first instance data in the form of a first generic object specification that corresponds to the first generic object class, the first instance data including attribute criteria, which are criteria that must be met by a specific user interface implementation that is selected using the first instance data, and hint criteria, which are criteria that are permitted but not required to be specified in the first instance data, and if specified, are permitted but not required to be met by a specific user interface implementation that is selected using the first instance data;

C. specifying in the application second instance data in the form of a second generic object specification that corresponds to the second generic object class, the second instance data including attribute criteria, which are criteria that must be met by a specific user interface implementation that is selected using the second instance data, and hint criteria, which are criteria that are permitted but not required to be specified in the second instance data, and if specified, are permitted but not required to be met by a specific user interface implementation that is selected using the second instance data;

D. providing in the computer system a first specific user interface toolbox and controller that is operable in the computer system to provide, for each of the first and second generic object classes, a respective first selection of multiple possible first specific user interface implementations;

E. providing in the computer system a second specific user interface toolbox and controller that is operable in the computer system to provide, for each of the first and second generic object classes, a respective second selection of multiple possible second specific user interface implementations;

F. providing in the computer system a first interpreter that corresponds to the first specific user interface toolbox and controller,

i. wherein the first interpreter is operable to select from the first specific user interface toolbox and controller a respective first specific user interface implementation for the first generic object class, such that a selected first specific user interface implementation satisfies both the attribute criteria and hint criteria specified for the respective first generic object class, except if no first specific user interface implementation satisfies both the attribute criteria and hint criteria specified for the first generic object class then the first interpreter being operable to select another first specific user interface implementation from the first selection that satisfies the attribute criteria but not all of the hint criteria that have been specified for the first generic object class, and

ii. wherein the first interpreter is operable to select from the first specific user interface toolbox and controller a respective first specific user interface implementation for the second generic object class, such that a selected first user interface implementation satisfies both the attribute criteria and hint criteria specified for the second generic object class, except if no first specific user interface implementation satisfies both the attribute criteria and hint criteria specified for the second generic object class then the first interpreter being operable to select another first specific user interface implementation from the first selection that satisfies the attribute criteria but not all of the hint criteria that have been specified for the second generic object class;

G. providing in the computer system a second interpreter that corresponds to the second specific user interface toolbox and controller,

i. wherein second interpreter is operable to select from the second specific user interface toolbox and controller a respective second specific user interface implementation for the first generic object class, such that a selected second specific user interface implementations satisfies both the attribute criteria and hint criteria of the respective first generic object class, except if no second specific user interface implementation satisfies both the attribute criteria and hint criteria of the first generic object class then the second interpreter being operable to select another second specific user interface implementation from the second selection that satisfies the attribute criteria but not all of the hint criteria that have been specified for the first generic object class, and

ii. wherein the second interpreter is operable to selected from the second specific user interface toolbox and controller a respective second specific user interface implementation for the second generic object class, such that a selected second user interface implementation satisfies both the attribute criteria and hint criteria of the second generic object class, except if no second specific user interface implementation satisfies both the attribute criteria and hint criteria of the second generic object class then the second interpreter being operable to select another second specific user interface implementation from the second selection that satisfies the attribute criteria but not all of the hint criteria that have been specified for the second generic object class;

H. selecting one of the first and second specific user interface toolbox and controllers;

i. in the event of the selection of the first specific user interface toolbox and controller, using the first interpreter to select respective first specific user interface implementations for the first generic object class and the second generic object class, and

ii. in the event of the selection of the second specific user interface toolbox and controller, using the second interpreter to select respective second specific user interface implementations for the first generic object class and the second generic object class.

* * * * *

US005867157A

United States Patent [19]

Goddard et al.

[11] Patent Number: 5,867,157

[45] Date of Patent: Feb. 2, 1999

[54] **GRAPHICAL INTERFACE METHOD, APPARATUS AND APPLICATION FOR CREATING AND MODIFYING A LIST OF VALUES WITH MULTIPLE COMPONENTS**

[75] Inventors: **Joan Stagaman Goddard**, Boulder, Colo.; **Minh Trong Vo**, Mountain View, Calif.

[73] Assignee: **International Business Machines Corporation**, Armonk, N.Y.

[21] Appl. No.: **696,753**

[22] Filed: **Aug. 14, 1996**

[51] **Int. Cl.**[6] **G06F 3/14**

[52] **U.S. Cl.** **345/333**; 345/340; 345/335

[58] **Field of Search** 345/333, 334, 345/335, 336, 339, 326, 347, 352, 354, 340; 707/505, 507

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,464,652	8/1984	Lapson et al.	345/165
5,001,654	3/1991	Winiger et al.	707/529
5,062,060	10/1991	Kolnick	345/339
5,072,412	12/1991	Henderson	345/346
5,095,512	3/1992	Roberts et al.	382/245
5,117,372	5/1992	Petty	345/335
5,119,476	6/1992	Texier	345/347
5,121,477	6/1992	Koopmans et al.	395/333

(List continued on next page.)

FOREIGN PATENT DOCUMENTS

2097540	12/1994	Canada .
0 587 394 A1	3/1994	European Pat. Off. .
0 62 728 A1	11/1994	European Pat. Off. .
04-361373	12/1992	Japan .
05-313845	11/1993	Japan .
06-4117	1/1994	Japan .
06-215095	8/1994	Japan .
07-129597	5/1995	Japan .

OTHER PUBLICATIONS

"Device Independent Graphics Using Dynamic Generic Operator Selection," *IBM Technical Disclosure Bulletin*, Apr. 1983, vol. 25, No. 11A, pp. 5477–5480.

"Error–Tolerant Dynamic Allocation of Command Processing Work Space," *IBM Technical Disclosure Bulletin*, Jun. 1984, vol. 27, No. 1B, pp. 584–586.

"Means for Computing the Max of a Set of Variables Distributed Across Many Processors," *IBM Technical Disclosure Bulletin*, Sep. 1990, vol. 33, No. 4, pp. 8–12.

"Graphical User Interface for the Distributed System Namespace," *IBM Technical Disclosure Bulletin*, Jul. 1992, vol. 35, No. 2, pp. 335–336.

"Graphical Query System," *IBM Technical Disclosure Bulletin*, Nov. 1993, vol. 36, No. 11, pp. 615–616.

"Configuration Data Set Build Batch Program," *IBM Technical Disclosure Bulletin*, Nov. 1993, vol. 36, No. 11, p. 571.

Self–Contained Reusable Programmed Components, *IBM Technical Disclosure Bulletin*, Jul. 1995, vol. 38, No. 7, pp. 283–285.

"IBM Printing Systems Manager for AIX Overview," *International Business Machines Corporation*, Second Edition, Feb. 1996.

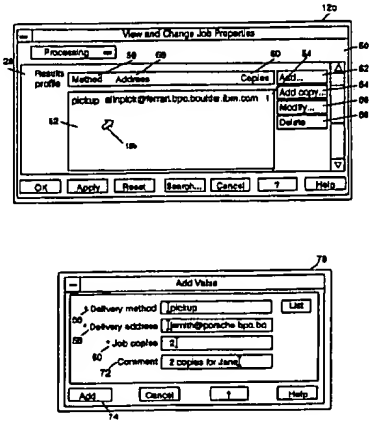
(List continued on next page.)

Primary Examiner—Matthew M. Kim
Assistant Examiner—Crescelle N. dela Torre
Attorney, Agent, or Firm—David W. Victor

[57] **ABSTRACT**

The invention is carried out in the following environment. The computer system has at least a visual operator interface, an operating system for operating applications within the computer system, and memory for storing at least part, preferably all, of an application. The present invention provides a method, apparatus, and application for constructing in a graphical user interface a list of values, with each value having multiple components. Also disclosed is a means for adding, modifying, and deleting values or components of values. Additionally disclosed is a means for copying selected values or components of values from one entry to another entry in the list of values.

8 Claims, 9 Drawing Sheets



U.S. PATENT DOCUMENTS

5,140,677	8/1992	Fleming et al.	345/348
5,140,678	8/1992	Torres	345/350
5,164,911	11/1992	Juran et al.	364/578
5,206,950	4/1993	Geary et al.	395/702
5,208,907	5/1993	Shelton et al.	707/505
5,228,123	7/1993	Heckel	345/334
5,233,687	8/1993	Henderson et al.	345/346
5,247,651	9/1993	Clarisse et al.	345/500
5,249,265	9/1993	Liang	345/356
5,255,359	10/1993	Ebbers et al.	345/433
5,276,901	1/1994	Howell et al.	707/9
5,287,447	2/1994	Miller et al.	345/342
5,307,451	4/1994	Clark	345/427
5,315,703	5/1994	Matheny et al.	345/507
5,317,687	5/1994	Torres	345/349
5,317,730	5/1994	Moore et al.	707/100
5,345,550	9/1994	Bloomfield	345/353
5,367,619	11/1994	Dipaolo et al.	707/506
5,371,844	12/1994	Andrew et al.	345/334
5,377,317	12/1994	Bates et al.	345/342
5,388,255	2/1995	Pytlik et al.	707/4
5,394,521	2/1995	Henderson et al.	345/346
5,404,439	4/1995	Moran et al.	345/328
5,410,695	4/1995	Frey et al.	395/680
5,410,704	4/1995	Norden-Paul et al.	395/671
5,412,776	5/1995	Bloomfield et al.	345/346
5,414,806	5/1995	Richards	345/435
5,416,900	5/1995	Blanchard et al.	345/346
5,418,950	5/1995	Li et al.	707/4
5,428,554	6/1995	Laskoski	364/550
5,428,776	6/1995	Rothfield	707/4
5,438,659	8/1995	Notess et al.	345/335
5,450,545	9/1995	Martin et al.	345/701
5,454,071	9/1995	Siverbrook et al.	345/441
5,454,106	9/1995	Burns et al.	707/4
5,459,825	10/1995	Anderson et al.	345/433
5,459,832	10/1995	Wolf et al.	345/342
5,463,724	10/1995	Anderson et al.	707/503
5,473,745	12/1995	Berry et al.	345/340
5,479,599	12/1995	Rockwell et al.	345/349
5,481,666	1/1996	Nguyen et al.	345/357
5,483,651	1/1996	Adams et al.	707/1
5,487,141	1/1996	Cain et al.	345/435
5,491,795	2/1996	Beaudet et al.	345/346
5,497,454	3/1996	Bates et al.	345/344
5,497,484	3/1996	Potter et al.	707/200
5,600,778	2/1997	Swanson et al.	345/333
5,603,034	2/1997	Swanson	345/333 X
5,774,667	6/1998	Garvey et al.	395/200.52

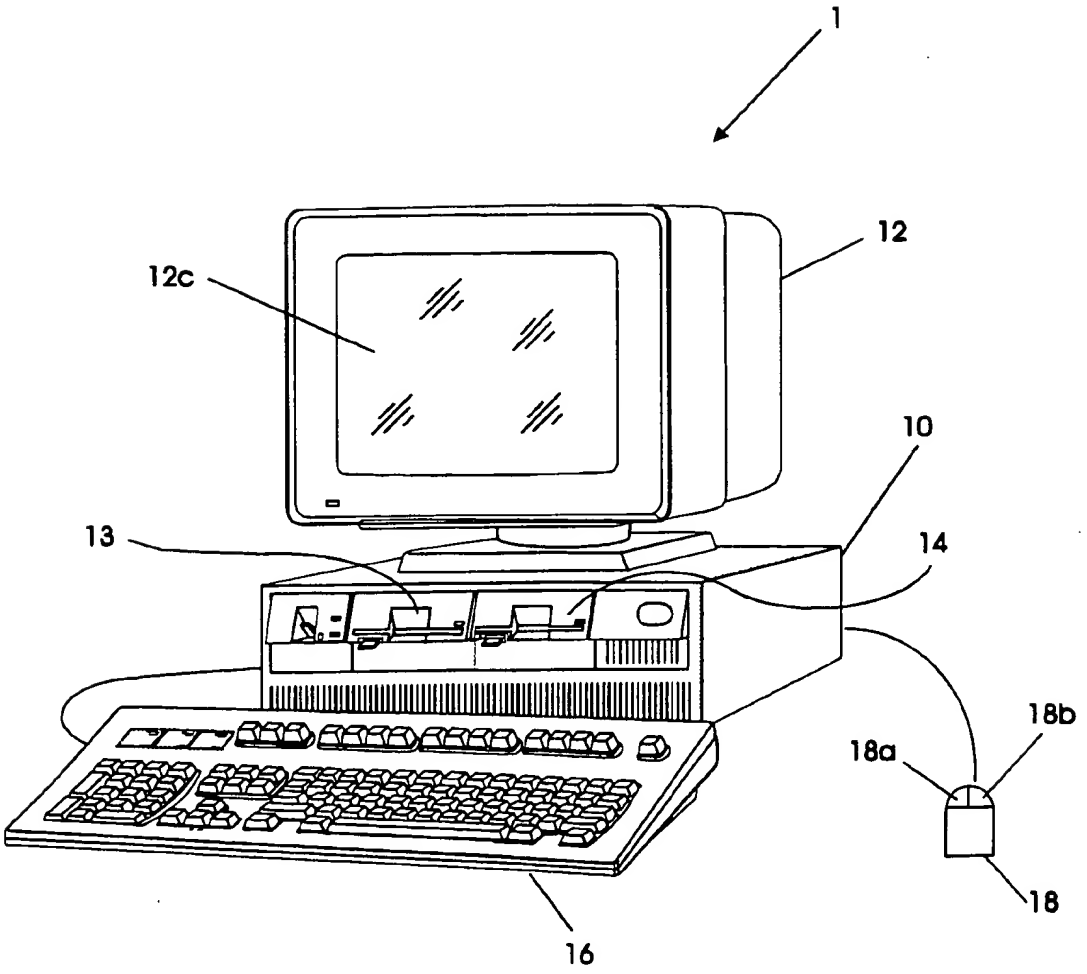
OTHER PUBLICATIONS

"IBM Printing Systems Manager for AIX Administrating," *International Business Machines Corporation,* 1995.

"Matching Three–Dimensional Objects Using a Relational Paradigm," *Pattern Recognition,* vol. 17, No. 4, pp. 385–405, 1984.

"A Multicolumn List–Box Container for OS/2," *Dr. Dobb's Journal,* May 1994, vol. 19, No. 5, pp. 90–94.

FIG. 1



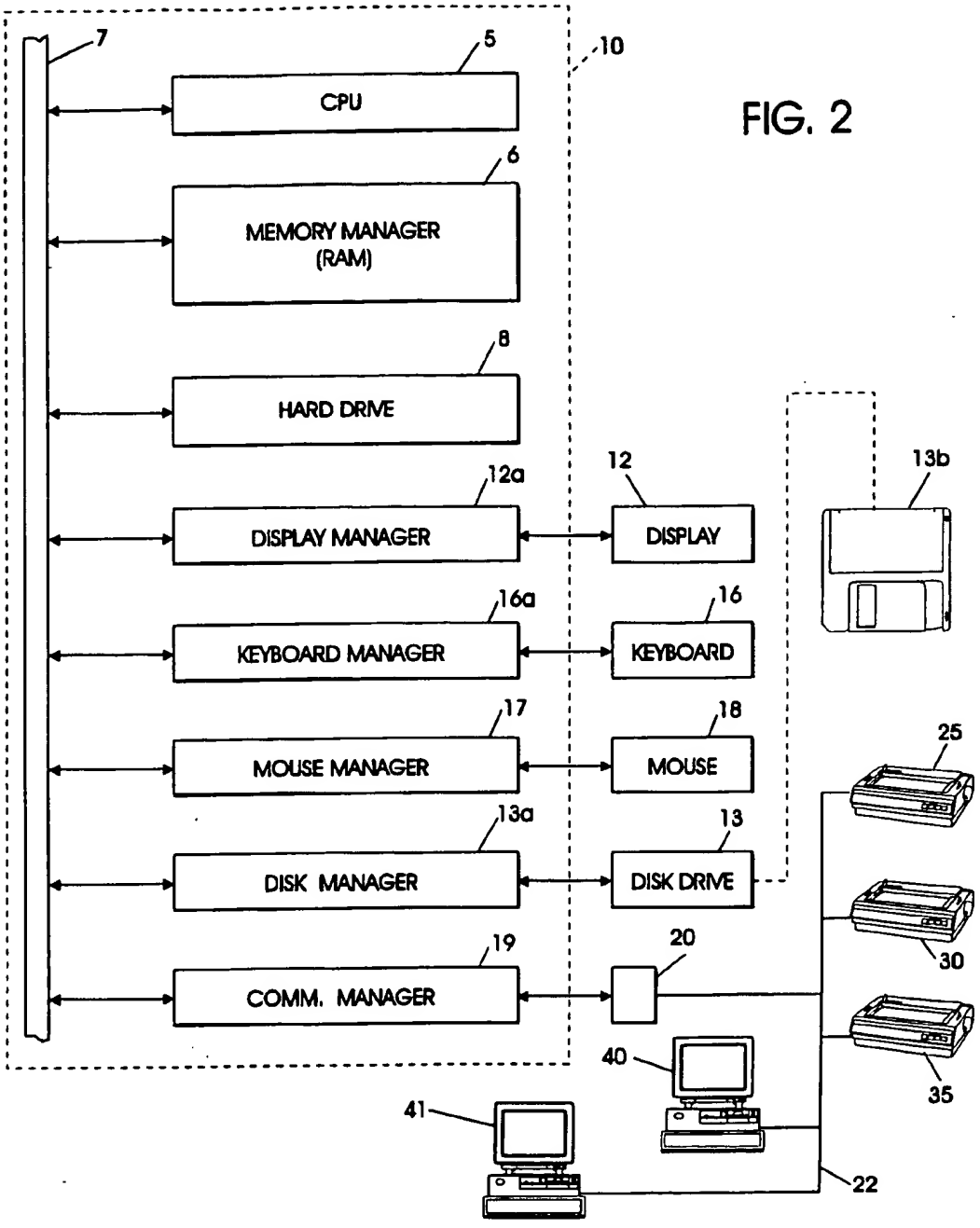

7
5
10
CPU
6
MEMORY MANAGER
(RAM)
8
HARD DRIVE
12
12a
13b
DISPLAY MANAGER
DISPLAY
16a
16
KEYBOARD MANAGER
KEYBOARD
17
18
25
MOUSE MANAGER
MOUSE
13a
13
DISK MANAGER
DISK DRIVE
19
20
COMM. MANAGER
30
40
35
41
22

FIG. 2

FIG. 3

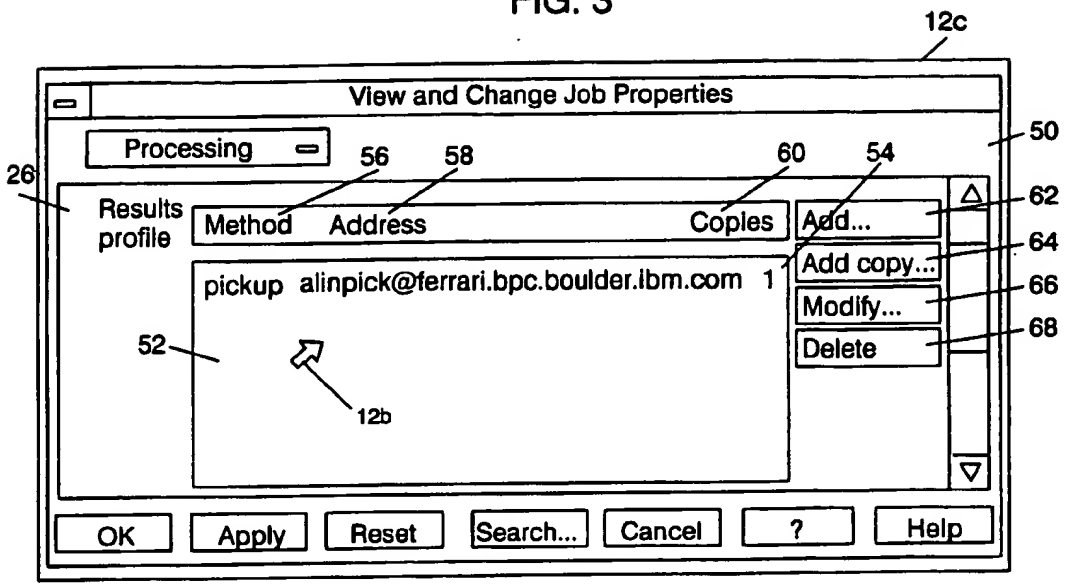

12c
View and Change Job Properties
Processing
50
26
56
58
60
54
Results profile
Method
Address
Copies
Add...
62
Add copy...
64
pickup alinpick@ferrari.bpc.boulder.ibm.com 1
Modify...
66
52
Delete
68
12b
OK
Apply
Reset
Search...
Cancel
?
Help

FIG. 4

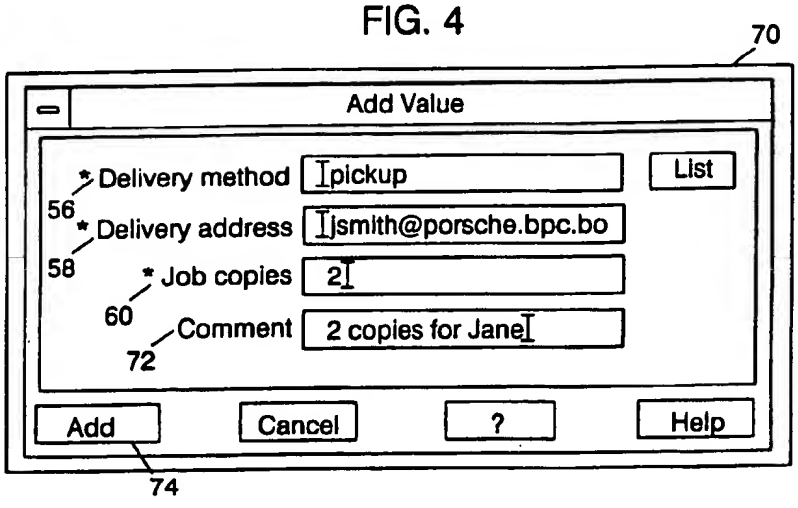

70
Add Value
* Delivery method
Ipickup
List
56
* Delivery address
Ijsmith@porsche.bpc.bo
58
* Job copies
2I
60
Comment
2 copies for JaneI
72
Add
Cancel
?
Help
74

FIG. 5

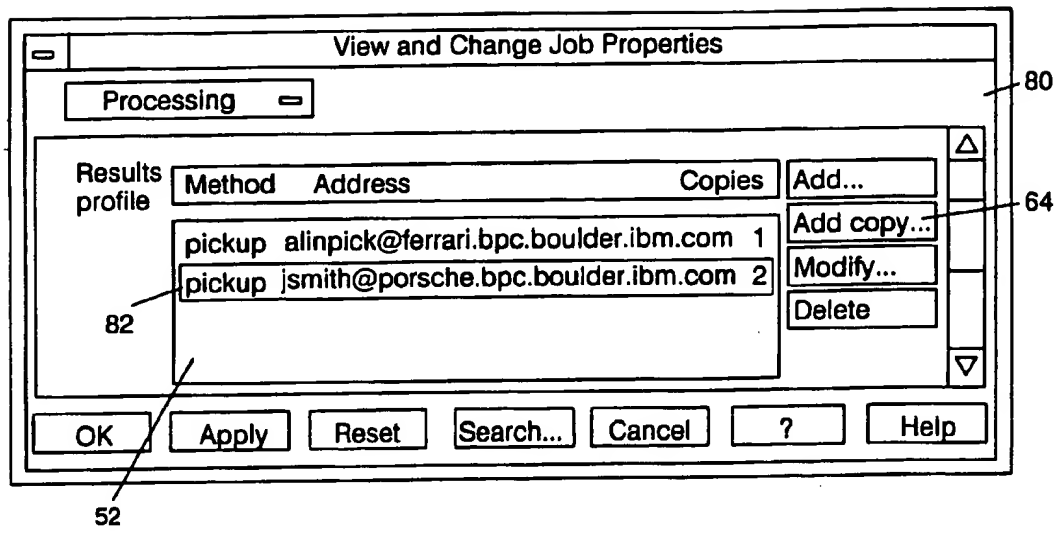

View and Change Job Properties
Processing
80
Results profile
Method Address Copies
Add...
Add copy...
64
Modify...
Delete
pickup alinpick@ferrari.bpc.boulder.ibm.com 1
pickup jsmith@porsche.bpc.boulder.ibm.com 2
82
OK
Apply
Reset
Search...
Cancel
?
Help
52

FIG. 6A

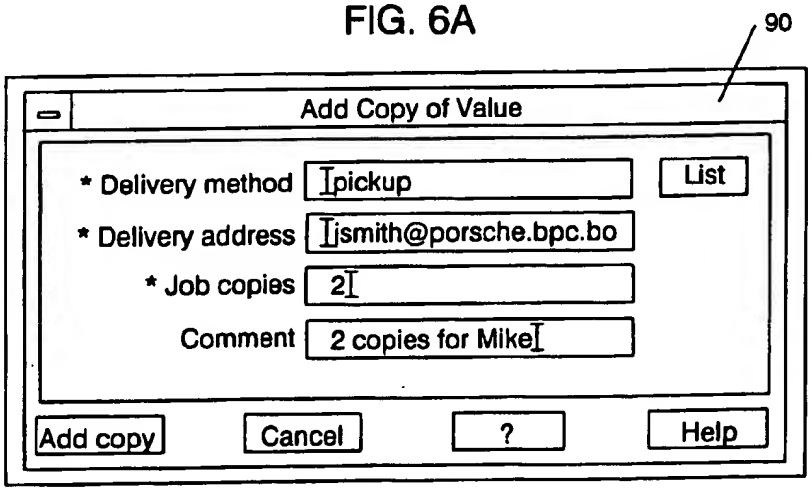

90
Add Copy of Value
* Delivery method
pickup
List
* Delivery address
jsmith@porsche.bpc.bo
* Job copies
2
Comment
2 copies for Mike
Add copy
Cancel
?
Help

FIG. 6B

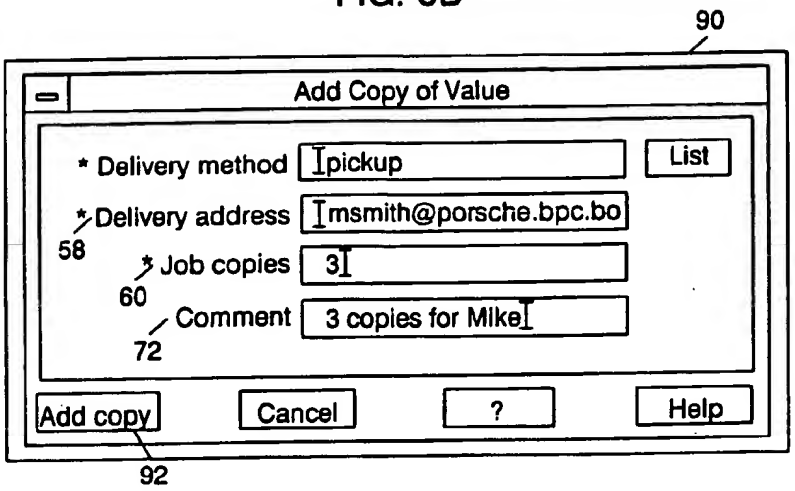

90
Add Copy of Value
* Delivery method
pickup
List
* Delivery address
msmith@porsche.bpc.bo
58
* Job copies
3
60
Comment
3 copies for Mike
72
Add copy
Cancel
?
Help
92

FIG. 7

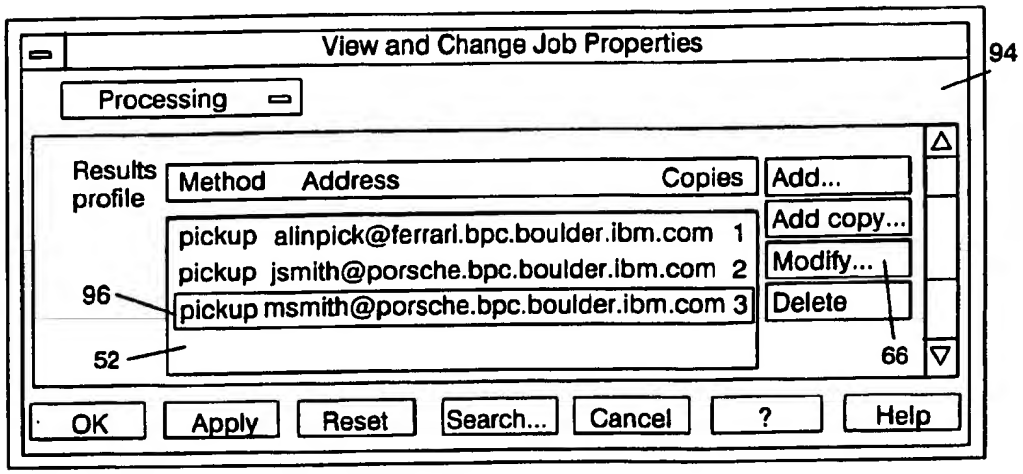

View and Change Job Properties
94
Processing
Results profile
Method Address Copies
Add...
Add copy...
Modify...
Delete
pickup alinpick@ferrari.bpc.boulder.ibm.com 1
pickup jsmith@porsche.bpc.boulder.ibm.com 2
96
pickup msmith@porsche.bpc.boulder.ibm.com 3
52
66
OK
Apply
Reset
Search...
Cancel
?
Help

FIG. 8A

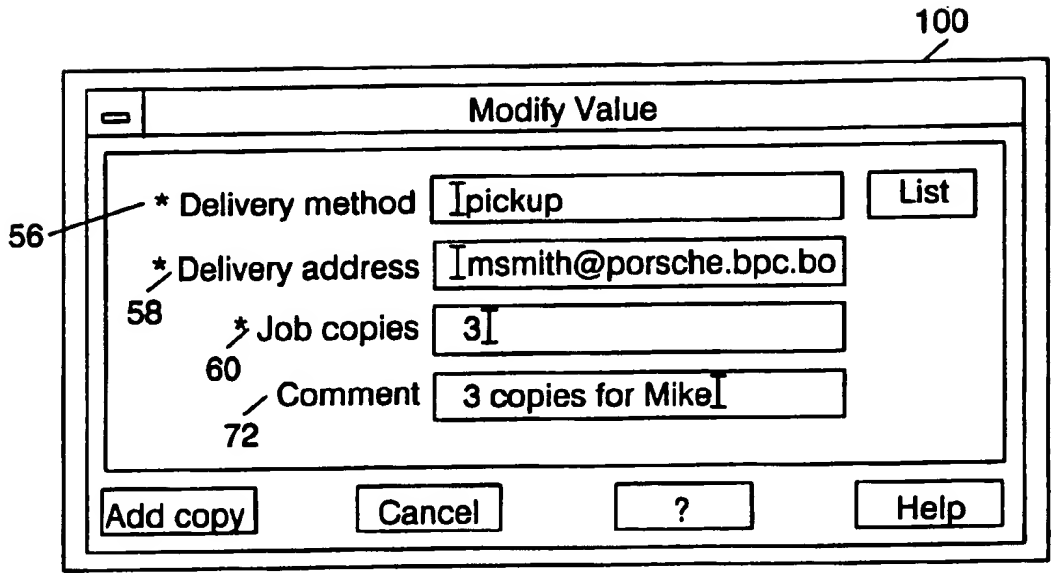

100
Modify Value
* Delivery method
pickup
List
56
* Delivery address
msmith@porsche.bpc.bo
58
* Job copies
3
60
Comment
3 copies for Mike
72
Add copy
Cancel
?
Help

FIG. 8B

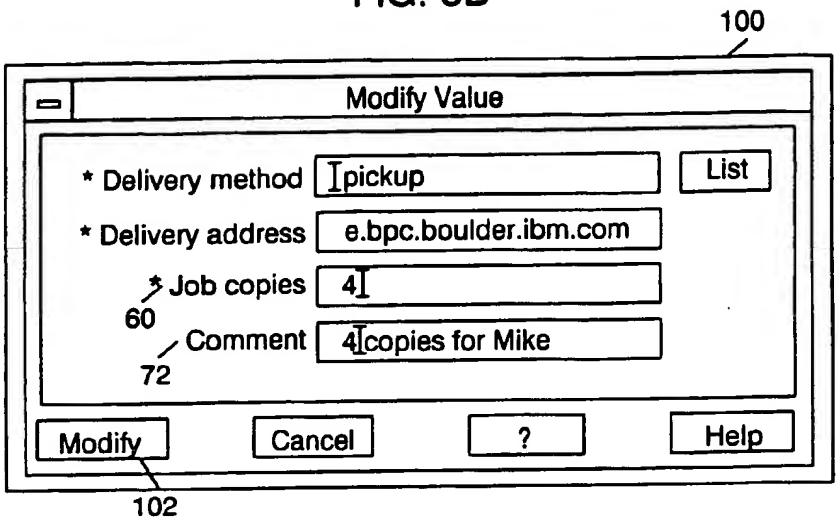

100
Modify Value
* Delivery method
pickup
List
* Delivery address
e.bpc.boulder.ibm.com
* Job copies
4
60
Comment
4copies for Mike
72
Modify
Cancel
?
Help
102

FIG. 9

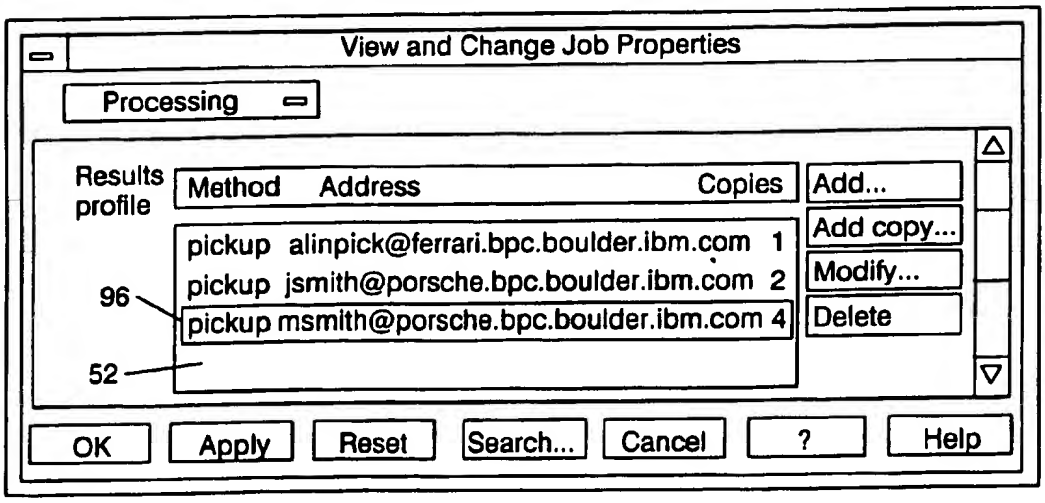
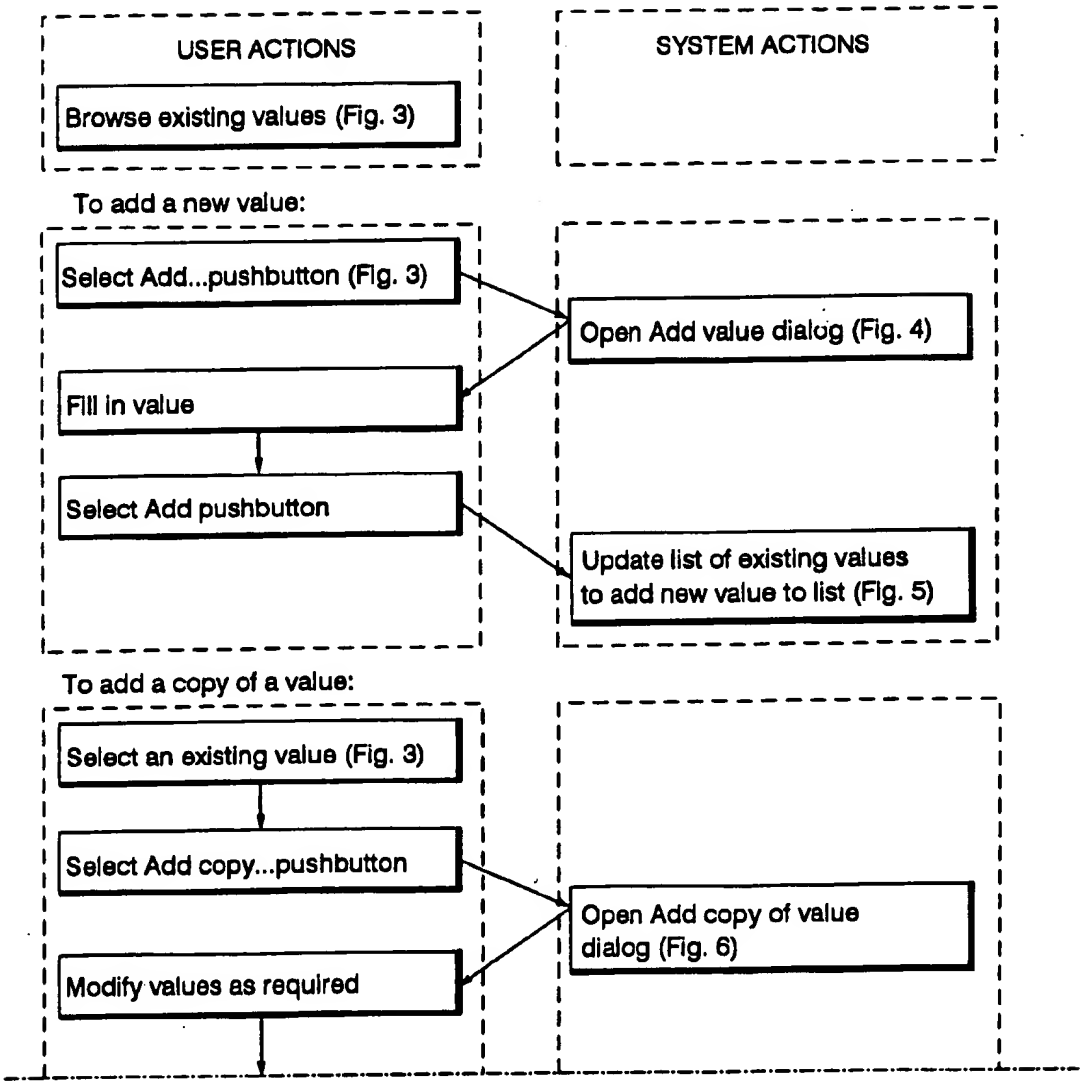

View and Change Job Properties
Processing
Results profile
Method Address Copies
Add...
pickup alinpick@ferrari.bpc.boulder.ibm.com 1
Add copy...
pickup jsmith@porsche.bpc.boulder.ibm.com 2
Modify...
96
pickup msmith@porsche.bpc.boulder.ibm.com 4
Delete
52
OK
Apply
Reset
Search...
Cancel
?
Help

FIG. 10A
FIG. 10B

FIG. 10

FIG. 10A



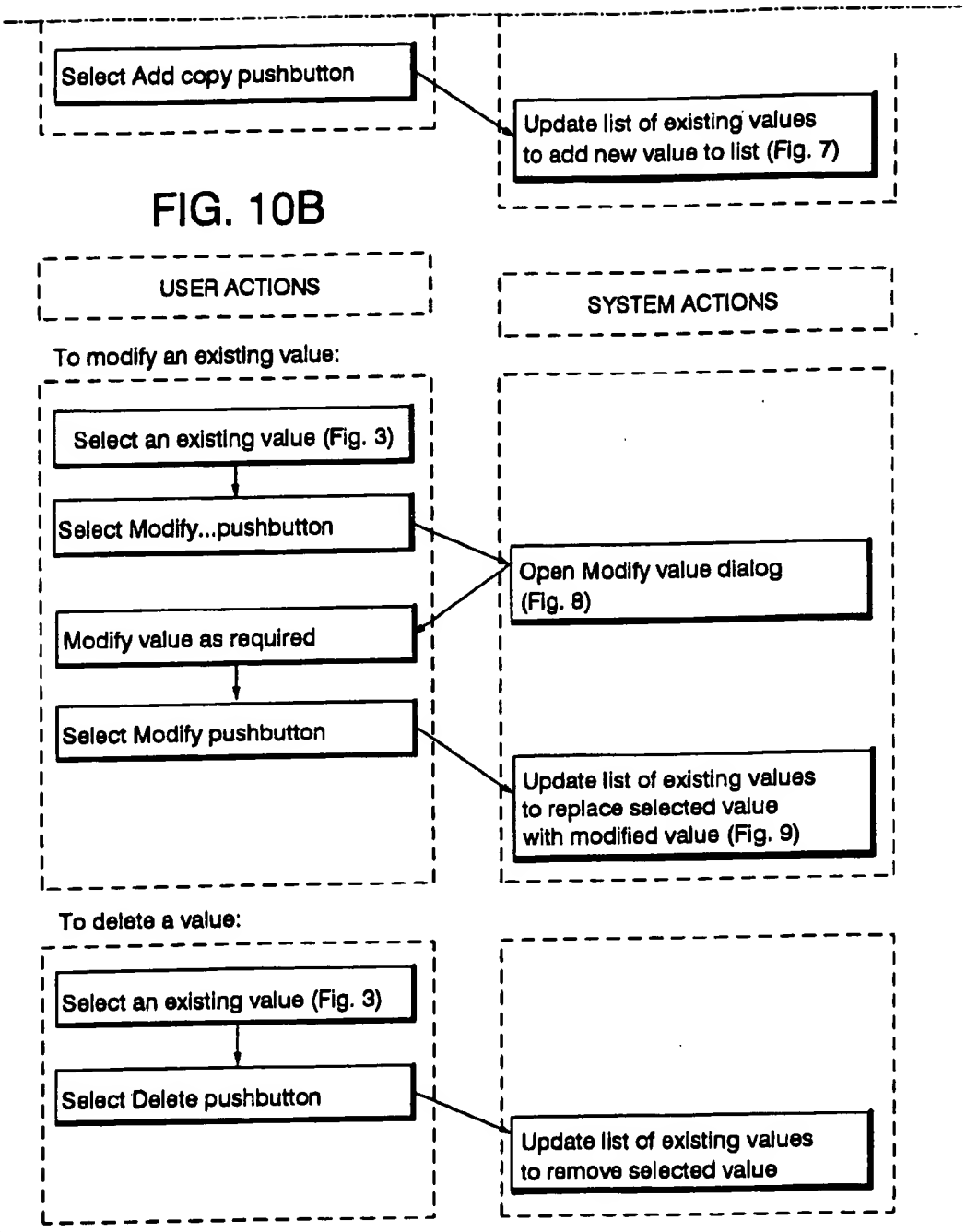

Select Add copy pushbutton
Update list of existing values to add new value to list (Fig. 7)
USER ACTIONS
SYSTEM ACTIONS
To modify an existing value:
Select an existing value (Fig. 3)
Select Modify...pushbutton
Open Modify value dialog (Fig. 8)
Modify value as required
Select Modify pushbutton
Update list of existing values to replace selected value with modified value (Fig. 9)
To delete a value:
Select an existing value (Fig. 3)
Select Delete pushbutton
Update list of existing values to remove selected value

FIG. 10B

GRAPHICAL INTERFACE METHOD, APPARATUS AND APPLICATION FOR CREATING AND MODIFYING A LIST OF VALUES WITH MULTIPLE COMPONENTS

BACKGROUND OF THE INVENTION AND STATE OF THE PRIOR ART

1. Field of the Invention

The present invention relates to graphical user interfaces. More particularly, the invention relates to a method, apparatus and application for creating and modifying a list of values, each value having multiple components.

2. Description of Related Art

In order to better understand the terms utilized in this patent application, a brief background definition section will be presented so that the reader will have a common understanding of the terms employed and associated with the present invention.

A "user interface" is a group of techniques and mechanisms that a person employs to interact with an object. The user interface is developed to fit the needs or requirements of the users who use the object. Commonly known user interfaces can include telephone push buttons or dials, or pushbuttons such as on a VCR or a television set remote. With a computer, many interfaces not only to allow the user to communicate with the computer but also allow the computer to communicate with the user. These would include (1) command-line user interfaces (i.e., user remembered commands which he/she enters, e.g. "C:>DIR" in which "DIR" is a typical DOS command entered at the "C" prompt); (2) menu-driven user interfaces which present an organized set of choices for the user, and (3) graphical user interfaces, ("GUI") in which the user points to and interacts with elements of the interface that are visible, for example by a "mouse" controlled arrow or cursor.

An example of a GUI user interface is that which is offered by International Business Machines Corporation (IBM) under the name "Common-User Access" ("CUA"). This GUI incorporates elements of object orientation (i.e., the user's focus is on objects and the concept of applications is hidden). Object orientation of the interfaces allow for an interconnection of the working environment in which each element, called an "object," can interact with every other object. The objects users require to perform their tasks and the objects used by the operating environment can work cooperatively in one seamless interface. With object oriented programming using a GUI, the boundaries that distinguish applications from operating systems are no longer apparent or relevant to the user.

In connection with this patent application, an "object" means any visual component of a user interface that a user can work with as a unit, independent of other items, to perform a task. By way of example, a spreadsheet, one cell in a spreadsheet, a bar chart, one bar in a bar chart, a report, a paragraph in a report, a database, one record in a database, and a printer are all objects. Each object can be represented by one or more graphic images, called "icons," with which a user interacts, much as a user interacts with objects in the real world. (NOTE: In the real world, an object might be an item that a person requires to perform work. As an example, an architect's objects might include a scale, T-square, and a sharp pencil, while an accountant's objects might include a ledger and a calculator.) However, it is not required that an object always be represented by an icon, and not all interaction is accomplished by way of icons.

While classification of objects may follow many different definitions, each class of objects has a primary purpose that separates it from the other classes. A class may be looked at as a group of objects that have similar behavior and information structures. In addition, each of the objects enumerated and defined below may contain other objects. There are three primary classes of objects. Each is discussed below.

(1) Container Object

This object holds other objects. Its principal purpose is to provide the user with a way to hold or group related objects for easy access or retrieval. An operating system, e.g. OS/2® (a trademark of IBM Corporation) or Windows® (a trademark of Microsoft Corporation), typically provides a general-purpose container, for example a folder or a program group—that holds any type of object, including other containers. For example, imagine a program group (or folder) labeled "PRIVATE FOLDER—ICONS". In the program group are three folder icons labeled "REPORTS", "PORTFOLIO" and "LETTERS". By selecting with a mouse or other pointing device the icon "PORTFOLIO", another window may open showing three more icons labeled "OIL PAINTINGS", "WATERCOLORS", and "PORTRAITS". In turn, selecting any of those three icons may open additional windows with further icons representing further subdivisions, or cross-references (e.g., "CUSTOMERS").

(2) Data Objects:

The principal purpose of a data object is to convey information. This information may be textual or graphical information or even audio or video information. For example, a business report displayed on the computer monitor may contain textual information concerning sales of "gadgets" over the past few years (text object) to all customers and also may contain a bar chart (graphic object) to pictorially depict, on the same monitor screen, the sales information.

(3) Device Objects:

The principal purpose of a device object is to provide a communication vehicle between the computer and another physical or logical object. Many times the device object represents a physical object in the real world. For example, a mouse object or icon can represent the user's pointing device, and a modem object can represent the user's modem, or a printer object or icon can represent the user's printer. Other device objects are purely logical, e.g. an out-basket icon representing outgoing electronic mail; a wastebasket object or icon representing a way the user may "trash" or dispose of other objects.

As can be seen from the foregoing, a class of objects may be defined as a description of the common characteristics of several objects, or a template or model which represents how the objects contained in the class are structured. While there are further ways in which to define objects and classes of objects, typically each class of objects will include similar attributes, the values of which the user will alter, modify, replace or remove from time to time. For a more complete discussion of objects, attributes, object oriented interfaces etc. see "Object Oriented Interface Design: IBM Common User Access" (published by Que, ISBN 1-56529-170-0).

The present invention relates primarily to data objects. In a graphical user interface, the user frequently desires to construct a list of values, with each value having multiple components. The components may be of different types, for example, numeric, boolean, or alphabetic. In addition to creating the list, the user typically wants to modify the list, once created, by adding, deleting, or modifying values or certain components of values. Moreover, it is useful for the user to be able to copy values from one entry to another.

There are no standard solutions to these needs in the field of graphical user interfaces. Of course a variety of database

products exist that provide various data management options. However, within graphical user interfaces there are no known solutions that adequately meet user needs.

SUMMARY OF THE INVENTION

In view of the above, it is a principal object of the present invention to provide a graphically oriented method, application and apparatus to construct a list of values, each value having multiple components.

Another object of the present invention is to permit the user to modify a list of values, each value having multiple components.

Yet another object of the present invention is to permit the user to copy selected values or components of values from one entry to another.

Still another object of the present invention is to provide an application which may be employed in a number of different computers, may be transported between different computers, and may be loaded into various computer environments.

The invention is carried out in the following environment. The computer system has at least a visual operator interface, an operating system for operating applications within the computer system, and memory for storing at least part, preferably all, of an application. The present invention provides a method, apparatus, and application for constructing in a graphical user interface a list of values, with each value having multiple components. Also disclosed is a means for adding, modifying, and deleting values or components of values. Additionally disclosed is a means for copying selected values or components of values from one entry to another entry in the list of values.

Other objects of the invention and a more complete understanding of the invention may be had by referring to the following description taken in conjunction with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWING(S)

FIG. **1** illustrates a typical desktop computer system which may be employed to practice the novel method and application of the present invention;

FIG. **2** is a block diagram illustrating a sample configuration of the computer system shown in FIG. **1**;

FIG. **3** is a typical window, in accordance with the present invention, showing a value in a list box with the value having multiple components;

FIG. **4** is a typical window showing how new values are added to the list of values;

FIG. **5** is a typical window showing the added values in the list box;

FIGS. **6**A and **6**B are typical windows showing the copying of a value already in the list box, and then modification of the copy;

FIG. **7** is a typical window showing the added copy, as modified, in the list box;

FIGS. **8**A and **8**B are typical windows showing the modification of existing values and components of values in the list box;

FIG. **9** is a typical window showing the modified value in the list box; and

FIG. **10** is a flow chart illustrating the method of the present invention, including FIG. **10**A and FIG. **10**B.

DESCRIPTION OF THE ILLUSTRATIVE EMBODIMENT(S)

Turning now to the drawings, and especially FIGS. **1** and **2**, FIG. **1** diagrammatically shows a computer system **1** which may be connected to a Local Area Network system (LAN **20**) as shown in FIG. **2**.

As shown in FIG. **1**, the computer system **1** comprises a main chassis **10**, a display means or monitor **12**, a connected keyboard **16** and a pointing device, in the present instance a mouse **18** which is operator controlled to move a pointer cursor **12***b* (shown in FIG. **3**) on the display or monitor screen **12***c*. As shown in FIG. **2**, the chassis **10** includes a central processing unit, or "CPU" **5**, a memory manager and associated random access memory, or "RAM" **6**, a fixed disk or hard drive **8** (which may include its associated disk controller), a display manager **12***a* which is connected externally to the chassis **10** of the display **12**; a keyboard manager **16***a*, which through flexible cable (not shown) is connected to the keyboard **16**; a mouse manager **17** (which in some instances may form part of the display manager **12***a*, and may be in the form of a software driver) for reading the motion of the mouse **18** and its control mouse buttons (MB) **18***a* and **18***b*, shown in FIG. **1**. A disk manager or controller **13***a* which controls the action of the disk drive **13** (and an optional drive such as a magneto-optical or CD ROM drive **14**) shown in FIG. **1**, rounds out most of the major elements of the computer system **1**.

The pointer element or cursor **12***b* can be moved over the display screen **12***c* by movement of the mouse **18**. The mouse buttons (MB) **18***a* and **18***b* give commands to the operating system, usually through a software mouse driver provided by the mouse manufacturer. With the first mouse button (MB) **18***a* the operator can select an element indicated on the display screen **12***c* using the pointer or cursor **12***b*, i.e., signify that an action subsequently to be performed is to be carried out on the data represented by the indicated element on the display screen **12***c*. The system normally gives some visual feedback to the operator to indicate the element selected, such as a change in color, or a blocking of the icon. The second mouse button (MB) **18***b* may be a menu button, if desired. Conventionally, when the operator presses button **18***b*, a selection menu or dialog with system commands will appear on the display screen **12***c*. The operator may select an icon or item from the selection menu or input information into the dialog box as appropriate using the cursor **12***b* and the first mouse button (MB) **18***a*. Some menu items, if selected, may call up another menu or submenu for the operator to continue the selection process.

The use of a mouse and selection menus is well known in the art, for example U.S. Pat. No. 4,464,652 to Lapson et al. describes a selection menu of the pull-down type in combination with a mouse. It should be recognized, of course, that other cursor pointing devices may be employed, for example a joystick, ball and socket, or cursor keys on the keyboard.

The foregoing devices (and software drivers therefore) within the chassis **10** communicate with one another via a bus **7**. To round out the computer system **1**, an operating system (not shown) must be employed. If the computer system is a typical IBM-based system, the operating system may be DOS-based and include a GUI interface such as contained in OS/2®, or WINDOWS®, or other operating system of choice. If the computer system is based upon RISC (reduced instruction set computer) architecture, then the operating system employed may be, in the instance of an IBM-based RISC architectured System/6000®, AIX. Alternatively, if the computer system **1** is a large host computer, such as a an IBM 3090, it may be running an operating system such as MVS or VM.

In the illustrated instance, the computer system **1** includes an I/O (Input/Output) manager or communications manager

19 (shown in FIG. 2) which serves to link the computer system for communications with the outside world such as to a systems printer, a modem or a LAN controller (such as a Token ring or ETHERNET or even through a modem employing SDLC) such as shown at 20 in FIG. 2. The LAN controller may be incorporated inside the computer system 1 or located externally as shown diagrammatically in FIG. 2, as desired. The LAN controller 20 may connect to other computer systems 40 and 41 as well as to other printers such as printers 25, 30 and 35 by communications cable 22 and the like. However the method and application of the present invention works equally well with multiple objects serviced by a single computer system.

Referring now to FIG. 3, the display screen 12c of the monitor 12 is shown with a window 50. As illustrated, a list box 52 contains a value 54 that has a number of components, 56, 58 and 60. Value 54 is one entry in the list box 52.

The components shown in FIG. 3, are method 56, address 58, and copies 60. In this example, there is an additional component, comment, which is scrolled to the right and therefore not visible in FIG. 3. From this example, it is clear that the components may be in a variety of formats. Components 56 and 58 are alphabetic, while component 60 is numeric. The components 56, 58, and 60 of the value 54 are listed horizontally in a row in the list box 52. Down the right hand side of window 50 are four pushbuttons 62, 64, 66, and 68. The Add Pushbutton 62 is selected to add a value (i.e., a new entry) to the list box 52. The Add Copy Pushbutton 64 is selected to copy a value from the list box 52 to create another entry in the list box 52. The Modify Pushbutton 66 is selected to modify an existing value in the list box 52. The Delete Pushbutton 68 is selected to delete an existing value in the list box 52. The operation of each of these pushbuttons is described in further detail below.

FIG. 4 shows a typical window 70 to demonstrate how values are added to the list box 52. When the user selects the Add Pushbutton 62 in the window 50 in FIG. 3, the window 70 appears. The window 70 lists on a separate line each component of a value in the list box 52. The components are delivery method 56, delivery address 58, job copies 60, and comment 72. Note that the window 70 lists the full name of each component, whereas the list box 52 lists an abbreviated name for each component 56, 58, and 60. In addition, window 70 has a comment box 72 for additional information about the value or its components. The information in the comment box 72 is scrolled to the right in the list box 52. To add a value to the list box 52, the user completes each component 56 to 60 on the window 70 and adds any additional comments in the comment box 72. The user then selects the Add Pushbutton 74 at the bottom of window 70. The window 70 then closes and, as shown in FIG. 5, the window 80 appears. Window 80 is identical to window 50, except that the value 82 added in window 70 is shown in the list box 52.

Assume now that the user desires to copy value 82 to create a third entry in list box 52. The user selects Add Copy Pushbutton 64. This causes window 90 to appear, as shown in FIG. 6A. As can be seen in FIG. 6A, each of the components of value 82 have been "pre-filled in" to each component field on window 90. If the user then desires to make certain modifications to the components this can be done without the user having to re-enter all of the information for the value 82.

As shown in FIG. 6B, minor modifications can be made to the delivery address component 58, the job copies component 60, and the comment box 72. When the modifications are complete, the user selects the Add Copy Pushbutton 92. This causes window 94 to appear, as shown in FIG. 7. As shown in FIG. 7, the new value 96 has been added to the list box 52.

Now suppose the user desires to modify value 96. The user selects the Modify Pushbutton 66. This causes the window 100 shown in FIG. 8A to appear. Window 100 shows each component 56, 58, and 60 of value 96. Each component 56, 58, and 60 and the comment box 72 is "pre-filled-in" to correspond with the information for value 96 showing in the list box 52 and as previously entered by the user. The user can make minor changes to the job copies component 60 as shown in FIG. 8B. The user also can change the comment box 72 as shown in FIG. 8B. Of course, other modifications are possible also. The modifications shown are examples only.

When the modifications are complete, the user selects the modify pushbutton 102 as shown in FIG. 8B. FIG. 9 shows list box 52 with value 96 modified as shown in FIG. 8B.

Although not illustrated in the Figures, the user also can delete a value in the list box 52. To delete a value, the delete pushbutton 68 is selected.

FIG. 10 shows the method of the present invention.

Although the invention has been described with a certain degree of particularity, it should be recognized that elements thereof may be altered by person(s) skilled in the art without departing from the spirit and scope of the invention. The invention is limited only by the following claims and their equivalents.

What is claimed is:

1. An application for creating a list of multiple values, wherein each value is comprised of multiple component values, wherein the application is controlled by a computer system having at least a visual operator interface, an operating system for controlling the operation of program applications within the computer system, and memory for storing a program application, the application comprising:

means for selecting a first window showing a list box for the values, wherein the displayed list box displays a list of values, the list box containing a heading for each component value;

means for selecting a first add pushbutton to view a second window in which each component may be filled in for an additional value to be added to the list box;

means for filling in each component value in the second window; and

means for selecting a second add pushbutton in the second window causing the first window to appear displaying the list box of values including the added value.

2. An application for copying a first value in a list box to create a second value in the list box, wherein a value is comprised of multiple component values, wherein the application is controlled by a computer system having at least a visual operator interface, an operating system for controlling the operation of program applications within the computer system, and memory for storing a program application, the application comprising:

means for selecting the first value from a first window showing the list box, wherein the displayed list box displays a list of values;

means for selecting a first add copy pushbutton second window showing the second value component value filled-in as for the selected and

means for selecting a second add copy pushb second window causing the first windov

displaying the list box of values including the second value as a new entry in the displayed list box.

3. An application for copying a first value with one or more component values in a list box to create a second value in the list box in accordance with claim 2 further comprising:

means for modifying the component values of the second value as desired prior to selecting the second add copy pushbutton.

4. An application for revising a value in a list box displaying multiple values, wherein a value is comprised of multiple components values, wherein the application is controlled by a computer system having at least a visual operator interface, an operating system for controlling the operation of program applications within the computer system, and memory for storing a program application, the application comprising:

means for selecting a first window displaying a list box, wherein the displayed list box displays a list of values,

means for selecting the value to be modified from the first window;

means for selecting a modify pushbutton to view a second window showing the components of the selected value;

means for modifying as desired the components shown in the second window; and

means for selecting a second modify pushbutton causing the first window to appear displaying the list box of values including the modified value.

5. A method for creating a list of multiple values, wherein each value is comprised of multiple component values, wherein the application is controlled by a computer system having at least a visual operator interface, an operating system for controlling the operation of program applications within the computer system, and memory for storing a program application, the application comprising:

selecting a first window showing a list box for the values, wherein the displayed list box displays a list of values, the list box containing a heading for each component;

selecting a first add pushbutton to view a second window in which each component may be filled in for an additional value to be added to the list box;

filling in each component in the second window; and

selecting a second add pushbutton in the second window causing the first window to appear displaying the list box of values including the additional value.

6. A method for copying a first value to create a second value, wherein the values are comprised of multiple component values, wherein the application is controlled by a computer system having at least a visual operator interface, an operating system for controlling the operation of program applications within the computer system, and memory for storing a program application, the application comprising:

selecting the first value from a first window showing the list box, wherein the displayed list box displays a list of values;

selecting a first add copy pushbutton to view a second window showing the second value with each component filled-in as for the selected first value; and

selecting a second add copy pushbutton in the second window causing the first window to appear displaying the list box of values including the second value as a new entry in the displayed list box.

7. A method for copying a first value with one or more component values in a list box to create a second value in the list box in accordance with claim 6 further comprising the steps of:

modifying the component values of the second value as desired prior to selecting the second add copy pushbutton.

8. A method for revising a value, wherein the value is comprised of multiple component values, wherein the application is controlled by a computer system having at least a visual operator interface, an operating system for controlling the operation of program applications within the computer system, and memory for storing a program application, the application comprising:

selecting a first window displaying a list box, wherein the displayed list box displays a list of values,

selecting the value to be modified from the first window;

selecting a modify pushbutton to view a second window showing the components of the selected value;

modifying as desired the components shown in the second window; and

selecting a second modify pushbutton causing the first window to appear displaying the list box of values including the modified value.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 5,867,157

DATED : February 2, 1999

INVENTOR(S) : Goddard et al.

It is certified that error appears in the above-identified patent and that said Letters Patent are hereby corrected as shown below:

Col. 7, line 3 and 4, delete "one or more" and insert multiple

Col 7, line 4, after "value", insert with multiple component values

Col. 7, line 11, delete "components" and insert component

Signed and Sealed this

Tenth Day of April, 2001

Attest:

Nicholas P. Godici

NICHOLAS P. GODICI

Attesting Officer

Acting Director of the United States Patent and Trademark Office